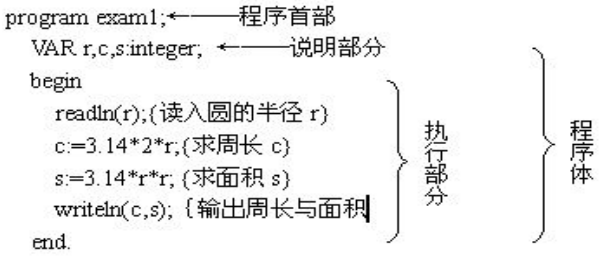
第一章　Pascal 语言基础知识

2.1 Pascal 程序基本组成

例 1.1 计算半径为 r 的圆的周长 C 和面积 S。



上述程序第一行称为程序首部。其中用花括号(注释可以用{ }或（* *）来表示)括起来的内容是注释，注释除了给人看，增加程序的可读性外，对程序编译和运行不起作用。一个程序可以包含多个出现在不同处注释，亦可无注释。程序第二行是常量说明。程序从 begin 到 end 都是执行（语句）部分。

(1)程序首部

例 1.1 的第一行称为程序首部。program 是保留字，接着是程序名（由你依据"标识符"规则自行定义），最后以分号表示程序首部结束，下面是程序体的开始。程序首部在一个 Turbo Pascal（仅在 Turbo Pascal 中有效）程序中并非必须出现，它是可选的。写上它仅起了文档作用。因此，在时间有限情况下，如果用 Turbo Pascal 编程完全可以省略程序首部。

(2)程序体

①说明部分

说明部分用于定义和说明程序中用到的数据，由单元说明、标号说明、常量说明、类型说明、变量说明、函数或过程说明组成，并且这些数据的说明次序必须按照以上次序。但是一个简单的 Turbo Pascal 程序也可以不包含说明部分，也就是说说明部分是可选的。

②执行部分

执行部分描述了程序要执行的操作。它必须以一个 Turbo Pascal 保留字 begin 开始，以保留字 end 后跟句点结束，其间是一些执行具体操作的语句，并且以分号作为语句之间的分隔符。begin 和 end 必须成对出现，这是一个 Turbo Pascal 程序所必须有的。紧跟 end 之后的句号表示执行部分的结束，也表示整个程序的结束。此后的任何语句都无效。Turbo Pascal 规定紧随 end 之前出现的分号允许省略。

(3)一个完全的 Pascal 程序结构

```
program 程序名;
   uses
       已知单元说明;
   label
       标号说明;
   const
      常量说明;
   type
      类型说明;
   var
      变量说明;
   function
      函数说明;
   procedure
      过程说明;
begin
   语句;
   语句;
   ……
```

语句

end.

2.2 Pascal 字符与符号

1.保留字(关键字)

所谓保留字是指在 Pascal 语言中具有特定的含义，你必须了解它的含义，以便于正确的使用，否则会造成错误。标准 Pascal 语言中的保留字一共有 35 个，Turbo Pascal 语言一共有 51 个。下面是 Pascal 语言的保留字:

and，array，begin，case，const，div，do，downto，else，end，file，for，funtion，goto，if，in，label，mod，nil，not，of，or，packed，procedure，program，record，repeat，set，then，to，type，until，var，while，with，exports，shr，string，asm，object，unit，constructor，implementation，destructor，uses，inherited，inline，interface，library，xor，shl

2.标识符

㈠标识符的定义：标识符就是以字母开头后面跟字母或数字的任意组合。标识符的长度仅由行的长度即 127 个字符所限制，但标准 Pascal 规定只有前 8 个字符是有效的，Turbo Pascal 规定标识符中的所有的字符均有效。并且大小写等效，不能包空格。可以用来标示常量、变量、程序、函数等。例如例 1.1 中的 exam1(程序名)，c、s、r(变量名)都是标识符。

㈡标识符的分类：

⑴标准标识符：指 Pascal 语言预先定义的标识符，具有特殊含义。以下列举了 Turbo Pascal 语言部分常用的标准标识符：

标准常量 False True Maxint

标准类型 Boolean Char Real Integer

标准函数 Abs Arctan Chr Cos Eof Eoln Exp Ln Odd Ord Pred Round Sin Sqr Sqrt Succ Trunc

标准过程 Dispose Get New Pack Page Put Read Readln Reset Rewrite Unpack Write Writeln

标准文件 Input Output

⑵用户自定义标识符：由你来根据需要定义。

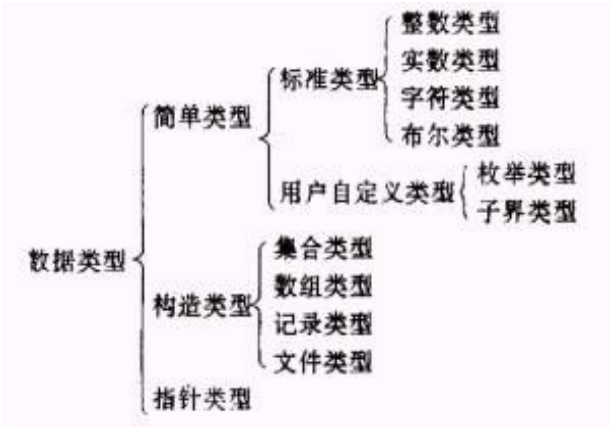
①选用的标识符不能和保留字相同。

②语法上允许预定义的标准标识符作为你定义的的标识符使用，但最好还是不要用。

2.3 Pascal 数据类型

数据是程序设计的一个重要内容，其重要特征----数据类型，确定了该数据的形、取值范围以及所能参与的运算。

Turbo Pascal 提供了丰富的数据类型，这些数据类型可以分为三大类：简单类型、构造类型和指针类型，其中简单类型可以分为标准类型（整型、实型、字符型和布尔型）和自定义类型（枚举型和子界型），构造类型可以分为数组类型、集合类型、记录类型和文件类型。这些数据类型中除了指针类型是动态数据类型外，其他的都是静态数据类型。在这些数据类型中简单类型都是有序类型，除了实型以外的简单类型都是顺序类型，所谓顺序类型就是他们的值不仅是有序的而且是有顺序号。



在这里主要介绍整型、实型、字符型和布尔型四种常用的数据类型。

1.整型

一个整型数据用来存放整数。Turbo Pascal 支持五种预定义整型，它们是 shortint（短整型）、integer（整型）、 longint（长整型）、 byte（字节型）和 word（字类型），Turbo Pascal 分

别用不同的名字作为他们的标识符。每一种类型规定了相应的整数取值范围以及所占用的内存字节数。

类型	数值范围	占字节数	格式
shortint（短整型）	-128..128	1	带符号 8 位
inteter（整型）	-32768..32767	2	带符号 16 位
longint（长整型）	-2147483648..2147483647	4	带符号 32 位
byte（字节型）	0..255	1	带符号 8 位
word（字类型）	0..65535	2	带符号 16 位

Turbo Pascal 规定了两个预定义整型常量标识符 maxint 和 maxlonint，他们各表示确定的常数值，maxint 为 32767， maxlonint 为 2147483647，他们的类型分别是 integer（整型）和 longint（长整型）。

2.实型

一个实型数据用来存放实数。Turbo Pascal 支持五种预定义实型，它们是 real（基本实型）、single（单精度实型）、double（双精度实型）、extended（扩展实型）、comp（装配实型），Turbo Pascal 分别用不同的名字作为他们的标识符。每一种类型规定了相应的实数取值范围、所占用的内存字节数以及它们所能达到的精度。

类型	数值范围	占字节数	有效位数
real（基本实型）	2.9e-39..1.7e38	6	11..12
single（单精度实型）	1.5e-45..3.4e38	4	7..8
double（双精度实型）	5.0e-324..1.7e308	8	15..16
extended（扩展实型）	3.4e-4932..1.1e4932	10	19..20
Comp（装配实型）	-2**63+1..2**63-1	8	19..20

Turbo Pascal 支持两种用于执行实型运算的代码生成模式：软件仿真模式和 80x87 浮点模式。除了 real 可以在软件仿真模式下直接运行以外，其他类型必须在 80x87 浮点模式下运行。

3.布尔型

一个布尔型数据用来存放逻辑值（布尔值）。布尔型的值只有两个：false 和 true，并且 false 的序号是 0，true 的序号是 1。false 和 true 都是预定义常数标识符，分别表示逻辑假和逻辑真。并且 true<false。boolean 是布尔型的标识符。

4.字符型

字符型用 char 作为标识符。字符型必须用单引号括起来，字母作为字符型时，大小写是不等价的，并且字符型只允许单引号中有一个字符，否则就是字符串。

2.4 常量与变量

1.常量

(1)常量：在某个程序的整个过程中其值不变的量。

(2)常量定义：常量定义出现在说明部分。它的语法格式是：

```
const
<常量标识符>=<常量>;
...
<常量标识符>=<常量>;
```

常量标识符的类型由定义它的常量的类型决定。例如：const a =12 隐含说明 a 是整型；const r=3.21 隐含说明 r 是实型.

(3)常量定义部分必须以保留字 const 开头，可以包含一个或几个常量定义，而且每个常量均以分号结束。

(4)Turbo Pascal 类型常量

类型常量，又称变量常数，它是 Turbo Pascal 的一个扩充特性。类型常量的定义与标准 Pascal 规定的常数定义和变量说明有所区别。类型常量定义的语法格式：

```
const
    <简单类型常量标识符>:简单类型=常数;
```

例如：

```
const
counter:integer=0;
```

flag:boolean=true;
index:0..100=0;

⑸常量的类型

㈠整型常量

整型常量采用我们平常使用的十进制整数表示。如 138,0,-512 等都是整型常量，而 18.或 18.0 都不是整型常量。

pascal 中有一个标准标识符 Maxint，它代表所使用的计算机系统允许的最大整型数，而最小的整型数即为－Maxint-1。Turbo Pascal 还定义了长整数常量 MaxLongInt，其值为 2147483647。

㈡实型常量

实型常量包括正实数、负实数和实数零。pascal 中表示实型常量的形式有两种。

①十进制表示法

这是人们日常使用的带小数点的表示方法。

如 0.0,-0.0,+5.61,-8.0,-6.050 等都是实型常量，而 0.,.37 都不是合法的实数形式。

②科学记数法

科学记数法是采用指数形式的表示方法，如 1.25×105 可表示成 1.25E+5。在科学记数法中，字母“E”表示 10 这个“底数”，而 E 之前为一个十进制表示的小数，称为尾数，E 之后必须为一个整数，称为“指数”。 如-1234.56E+26 ， +0.268E-5 ， 1E5 是合法形式，而.34E12, 2.E5 ，E5 ,E,1.2E+0.5 都不是合法形式的实数。

无论实数是用十进制表示法还是科学表示法，它们在计算机内的表示形式是一样的，总是用浮点方式存储。

和整数相比，实数能表示的范围大得多，但值得注意的是实数的运算比整数的运算速度慢且无法像整数那样精确表示，只能近似表示。

㈢字符常量

在 Pascal 语言中，字符常量是由单个字符组成，所有字符来自 ASCII 字符集，共有 256 个字符。在程序中，通常用一对单引号将单个字符括起来表示一个字符常量。如：’a’，’A’，’0’等。特殊地，对于单引号字符，则要表示成’’’’。对于 ASCII 字符集中，按每个字符在字符集中的位置，将每个字符编号为 0－255，编号称为对应字符的序号。

㈣布尔常量

布尔型常量仅有两个值，真和假，分别用标准常量名 true 和 false 表示。它们的序号分别为 1 和 0。

㈤符号常量

一个常量即可以直接用字面形式表示（称为直接常量， 如 124,156.8），也可以用一个标识符来代表一个常量，称为"符号常量"。但符号常量必须在程序中的说明部分定义，也就是说先定义，后使用。

定义符号常量的一般格式:

CONST

<常量标识符>=<常量>

说明：常量说明部分以关键字 const 开头，后面的标识符为常量标识符，其中"="号后的常量为整数、实数、字符、 字符串。而且，在常量说明部分可以将几个常量说明成符号常量，共用一个关键字"const"。例如:

```
program ex;
const
     pi=3.14159;          } 常量说明
     zero=0;              }
var r,c,s:real;
begin
  write('Enter r=');readln(r);
  c:=2*pi*r;
  s:=pi*r*r;
  writeln('c=',c);
  writeln('s=',s);
end.
```

则在本程序中 pi 和 zero 作为符号常量,分别代表实数 3.14159 和整数 0。也就是说，常量说明部分既定义了常量名及其值，又隐含定义了常量的类型。

关于符号常量，应注意下列几点：

①符号常量一经定义，在程序的执行部分就只能使用该常量标识符，而不能修改其值。

②使用符号常量比直接用数值更能体现“见名知义”的原则，也便于修改参数，故一个较好的程序中，应尽量使用符号常量，在执行部分基本上不出现直接常量。

2.变量

变量：在某个程序中的运行过程中其值可以发生改变的量。

变量有三个要素是：变量名、变量类型、变量值。

一个程序中可能要使用到若干个变量，为了区别不同的变量，必须给每个变量（存贮单元）取一个名（称为变量名），该变量（存贮单元）存放的值称为变量的值，变量中能够存放值的类型为变量的类型。

⑴变量名

用一个合法的标识符代表一个变量。如 n,m,rot,total 等都是合法变量名。在程序中用到的变量必须在说明部分加以说明，变量名应遵循自定义标识符的命名规则，并注意“见名知义”的原则，即用一些有意义的单词作为变量名。

“自定义标识符”的命名规则为：自定义标识符必须以字母（包含下划线"_"）开头，后面的字符可以是字母或数字。

⑵变量的类型

常量是有类型的数据，变量在某一固定时刻用来存放一个常量，因此也应有相应的类型。 如整型变量用来存放整数，实型变量用来存放实数。

⑶变量说明

在程序中若要使用变量，变量的名称及类型在程序的变量说明部分加以定义，变量的值则在程序的执行部分中才能赋给。

变量说明的一般格式：

VAR

<变量标识符>[,<变量标识符>]:<类型>;

(中括号内部分表示可省,下同)

其中 VAR 是 pascal 保留字,表示开始一个变量说明段, 每个变量标识符或由逗号隔开的多个变量标识, 必须在它的冒号后面说明成同一类型。一个程序中，可以说明许多不同类型的变量，每种类型变量之间用分号隔开，共用一个 VAR 符号。

例如:

```
var
age,day:integer;
amount,average:real;
```

其中，Integer(整型)、Real(实型)是标准标识符， 它们是"类型标识符"，代表了确定的类型，如 age 和 day 被定义为整型变量，amount 和 average 被定义为实型变量。一旦定义了变量，就确定了它的类型，也就是说，就确定了该变量的取值范围和对该变量所能进行的运算。

2.5 标准函数

1.算术函数

函数标识符	自变量类型	意义	结果类型
abs	整型、实型	绝对值	同自变量
arctan	整型、实型	反正切	实型
cos	整型、实型	余弦	实型
exp	整型、实型	指数	实型
frac	整型、实型	小数部分	实型
int	整型、实型	整数部分	实型
ln	整型、实型	自然对数	实型
pi	无自变量	圆周率	实型
sin	整型、实型	正弦	实型
sqr	整型、实型	平方	同自变量

sqrt	整型、实型	平方根	实型

例：abs(-4)=4 abs(-7.49)=7.49
arctan(0)=0.0 sin(pi)=0.0 cos(pi)=-1.0
frac(-3.71)=-0.71 int(-3.71)=-3.0
sqr(4)=16 sqrt(4)=2

2.标量函数

函数标识符	自变量类型	意义	结果类型
odd	整型	判断奇数	布尔型
pred	离散类型	求前趋	同自变量
succ	离散类型	求后继	同自变量

例：odd(1000)=false odd(3)=true
pred(2000)=1999 pred('x')='w'
succ(2000)=2001 succ('x')='y'

3.转换函数

函数标识符	自变量类型	意义	结果类型
chr	byte 型	自量对应的字符	字符型
ord	离散类型	自量对应的序号	longint
round	实型	四舍五入	longint
trunc	实型	截断取整	longint

4.杂类函数

函数标识符	自变量类型	意义	结果类型
random	无自变量	[0,1]之间的随机实数	real
random	word	[0,自变量]之间的随机整数	wird
randomize	无自变量	用一随机值初始化内部随机数产生器	longint
upcase	字符型	使小写英文字母变为大写	字符型

2.6　运算符和表达式

1.运算符和优先级

(1)运算符

①算术运算符

运算符	运算	运算对象	结果类型
+	加	整型、实型	只要有一个运算对象是实型，结果就是实型，如果全部的运算对象都是整型并且运算不是除法，则结果为整型，若运算是除法，则结果是实型
-	减	整型、实型	
*	乘	整型、实型	
/	除	整型、实型	
div	整除	整型	整型
mod	取余	整型	整型

②逻辑运算符

运算符	运算	运算对象	结果类型
not	逻辑非	布尔型	布尔型
and	逻辑与	布尔型	布尔型
or	逻辑或	布尔型	布尔型
xor	逻辑异或	布尔型	布尔型

③关系运算符

运算符	运算	运算对象	结果类型
=	等于	简单类型	布尔型
<>	不等于	简单类型	布尔型
<	小于	简单类型	布尔型

>	大于	简单类型	布尔型
<=	小于等于	简单类型	布尔型
>=	大于等于	简单类型	布尔型

(2)优先级

运算符	优先级
not	1(高)
*,/,div,mod,and	2
xor,+,-,or	3
in,=,<>,>=,<=,<>	4（低）

2.表达式 返回

(1)算术表达式：算术表达式是由算术运算符连接常量、变量、函数的式子。算术表达式中各个运算符的次序为： ( )-->函数-->*,/,div,mod-->+,1

(2)布尔表达式：Turbo Pascal 提供给布尔表达式以下基本操作：逻辑运算和关系运算。

第二章 顺序结构程序设计

3.1 赋值语句

赋值语句是最简单的语句，其一般形式为：<变量名>:=<表达式>

赋值语句的作用是计算表达式的值，并赋给变量。对于任何一个变量必须首先赋值，然后才能引用，否则，未赋初值的变量将以一个随机值参与运算。另外，赋值号两边的类型必须相同，但表达式值为整数时，它可自动化为实型后赋给该实型变量，即符合赋值相容。

注意：“:=”赋值号与“=”等号的含义不同，赋值号是将其右边内容赋值给左边的变量，使左边变量具有一个新值。而等号表示将两边的内容进行比较，结果为相等或不相等。赋值号的左边不能是表达式或常数，只能是变量名或自定义函数名，等号左边无此限制。

例如：X:=X+1 表示将 X 的值加 1 之后赋给 X，从而 X 具有一个新的值，Y+X:=X+1 是错误的赋值语句。

例：关于赋值的例子：

```
program example;
 var a,b:integer;
begin
 a:=3;b:=2;
 writeln(a);
 writeln(b);
 a:=a+b;
 writeln(a);
 writeln(b);
 b:=a-b;
 writeln(a);
 writeln(b);
 a:=a-b;
 writeln(a);
 writeln(b);
 readln
end.
```

3.2 输入语句（读语句）

在程序中变量获得一个确定的值，固然可以用赋值语句，但是如果需要赋值的变量较多，或变量的值经常变化，则使用本节介绍的输入语句——读语句，将更为方便。读语句是在程序运行时由用户给变量提供数据的一种很灵活的输入动作，它有两种格式：

1.读语句的一般格式：

```
read(<变量名表>);
readln[(<变量名表>)];
readln;
```

其中变量名表是用逗号隔开的若干个变量名组成的。

功能：从标准输入文件中读入数据,并依次赋给相应的变量。

说明：

①read 和 readln 是标准过程名,它们是标准标识符。

②执行到 read 或 readln 语句时，系统处于等待状态，等待用户从键盘上输入数据，系统根据变量的数据类型的语法要求判断输入的字符是否合法。如执行 read(a)语句，a 是整型变量，则输入的字符为数字字符时是合法的，当输入结束时，则自动将刚接受的一串数字
字符转换为整数赋给变量 a。

③在输入数值型（整型或实型）数据时，数据间要用空格或回车分隔开各个数据，输入足够个数的数据，否则仍要继续等待输入，但最后一定要有回车，表示该输入行结束，直到数据足够，该读语句执行结束，程序继续运行。

例 1：设 a、b、c 为整型变量，需将它们的值分别赋以 10,20,30,写出对应下列语句的所有可能输入格式。

Read(a,b,c);

解：根据③,即可列出所有可能输入格式

(a)10□20□30←┘

(b)10□20←┘

30←┘

(c)10←┘

20□30←┘

(d)10←┘

20←┘

30←┘

其中"←┘"表示回车键。下同。

④read 语句与 readln 语句的第一个区别是：

read 语句是一个接一个地读数据，在执行完本 Read 语句(读完本语句中变量所需的数据)后,下一个读语句接着从该数据输入行中继续读数据，也就是说，不换行。

例 2:

Read(a,b);

Read(c,d);

Read(e);

如果输入数据行如下:

1□2□3□4□5□6□←┘

则 a,b,c,d,e 的值分别为 1,2,3,4,5,如果后面无读语句则数据 6 是多余的，这是允许的。

Readln 则不同，在读完本 Readln 语句中变量所需的数据后, 该数据行中剩余的数据多余无用，或者说，在读完本 Readln 语句中变量所需数据后，一定要读到一个回车，否则多余的数据无用。

例 3：设要达到例 1 同样的目的，但语句改为:

readln(a,b);readln(c)

则例 1 中的 4 种输入格式只有(b)(d)是有效的.

⑤readln 语句与 read 语句的第二个区别是:

read 后一定要有参数表,而 readln 可以不带参数表,即可以没有任何输入项, 只是等待读入一个换行符(回车)。经常用于暂停程序的运行，直到输入一个回车。

例 4 ：设有下列语句:

read(a,b,c);

readln(d,e);

readln;

readln(f,g);

其中,所有变量均为整型。再设输入的数据如下：

1□2←┘
3□4□5□6□7□8←┘
9□10←┘
11←┘
12□13←┘
列表给出每个变量的值。

分析：

可以假想有一"数据位置指针"，每读一个数据后，指针后移到该数据之后，每执行一个 readln 语句后，指针移到下一个数据行的开头。

各变量的值如下表所示。

变量名	a	b	c	d	e	f	g
值	1	2	3	4	5	11	12

⑥为了避免可能出现的错误，建议在程序中按下列原则使用读语句：

(A)如果没有特殊需要,在一个程序中尽量避免混合使用 read 语句和 readln 语句;

(B)尽量用 readln 语句来输入数据, 一个数据行对应一个 readln 语句;

(C)由于执行 read 或 readln 语句时, 系统不会提供任何提示信息,

因此,编程时最好在 readln 语句之前加以适当提示,例如:

```
write('Input a,b,c:');
readln(a,b,c);
```

在执行时,屏幕上显示:

```
Input a,b,c:■
```

其中,"■"为光标。执行 readln 语句后,系统处于待待输入状态, 只有输入了所需数据后才继续往下执行。

3.3 输出语句

输出是将内存中的数据送到外设的过程。Turbo Pascal 的输出语句有两种形式：

```
write(<输出项表>)
writeln(<输出项表>)
writeln
```

其中<输出项表>是一串用逗号分隔的常量、变量、函数名、表达式或字符串。如果是变量、函数名、表达式，则将其计算结果输出；如果是常量或字符串，则直接输出其值。

write 和 writeln 的区别在于：write 语句是输出项输出后，不换行，光标停留在最后一项后，writeln 语句按项输出后，自动换行，光标则停留在下一行的开始位置。

writeln 语句允许不含有输出项，即仅 writeln;表示换行。

Turbo Pascal 语言把输出项的数据显示占用的宽度称为域宽，你可以根据输出格式的要求在输出语句中自动定义每个输出项的宽度。定义宽度时分为单域宽和双域宽。

（1）隐含的输出格式

pascal 语言为整型量、实型量、布尔型量和字符串（用一对单引号括起来的字符序列）规定了每种数据所占的宽度（即一个数据占几列），一个数据所占的宽度称为“场宽”或“字段宽”。系统给出的隐含场宽称为标准场宽。每一种 pascal 版本给定的标准场宽不尽相同。

下表给出标准 pascal 和 pc 机上两种 pascal 版所规定的标准场宽。

数据类型	标准 pascal	Turbo pascal
integer	10	实际长度
real	22	17
布尔型	10	4 或 5
字符串	串长	串长

在 Turbo Pascal 系统中, 对于整型字符串的输出都是按数据本身长度输出,对于布尔型数据 (只有 True 和 False 两种值），TRUE 为 4 列，FALSE 为 5 列，一律采用大写输出。而 real 型数据的输出时，则按 17 列输出，其中第一列为符号位，正号不显示，后四位为"E±nn"， 中间的 12 列为尾数部分。如：

```
writeln(sqrt(75));
```

则输出□8.6602540379E+00。

而 writeln(sqrt(81));

则输出□9.0000000000E+00。

有时，在程序中往往根据实际情况，需要自己定义场宽。

（2）单域宽输出格式：

writeln(I：n)

在n个字符宽的输出域上按右对齐方式输出I的值,若n大于I的实际位数,则在I值前面补(n-I的实际位数)个空格。若I的实际位数大于n，则自动突破限制。n必须是整数。

如write(1234:8);write('abcdef':12)

输出结果:

□□□□1234□□□□□□abcdef

对于标准实型数据指定单场宽时,如果场宽大于标准场宽时,右靠齐按标准场宽格式输出17位,左留空格。若场宽小于标准场宽时，第一位仍为符号位，最后四位仍为“E±nn”，中间部分为尾数显示部分。如果指定的宽度小于8位，则数据按8位格式“*.*E±nn ”输 出。

（3）双域宽输出格式：

如果输出项是实数时,如果希望输出的实数不用科学记数法输出,而用小数形式输出,可以用指定双场宽方法输出。

输出格式： writeln(a：m：n)

双域宽主要用于实型数据的输出。m的用法同上。在m个字符宽的输出域上按右队齐方式用小数点形式输出a的数值，n是小数点后的位数。原来的数据按该该格式指定的小数位数四舍五入。若n=0 ，则不输出小数部分和小数点，原数据四舍五入取整。n，m必须是整 数。

例如：整型变量I为15，实型变量J为-35.23，下列输出语句的参数格式及执行结果 为：

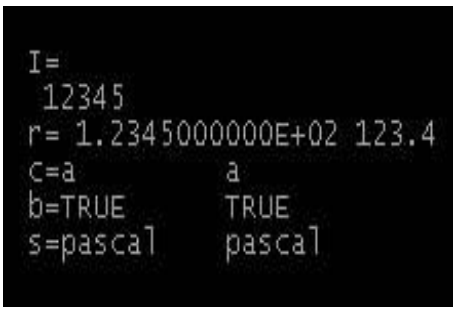
write(I:5); 15（15前面有三个空格）

write('ABC':5); ABC（ABC前面有两个空格）

write(J:8:2); -35.23（-35.23前面有两个空格）

例：输出语句的例子

```
program shuchu;
  const
    s='pascal';
  var
    i:integer;
    r:real;
    c:char;
    b:boolean;
begin
  i:=12345;
  r:=123.45
  c:='a';
  b:=true;
  writeln('i=');
  writeln(i:6);
  writeln('r=',r,r:6:1);
  writeln('c=',c,c:10);
writeln('b=',b,b:10);
writeln('s=',s,s:10);
readln;
  end.
```



3.4 复合语句

复合语句是由若干语句组成的序列，语句之间用分号"；"隔开，并且以 begin 和 end 括起来，作为一条语句。复合语句的一般形式：

```
begin
  语句 1;
  语句 2;
  ……
  语句 n;
end
```

注意：PASCAL 把复合语句视作一个语句。复合语句可用在 PASCAL 允许使用普通语句的任何地方。

例：变量值的交换

```
program jiaohuan;
  var
    a,b,t:integer;
begin
  a:=10;b:=20;
  begin
    t:=a;
    a:=b;
    b:=t;
  end;
  writeln('a=',a,'b=',b)
end.
```

第三章 选择结构程序设计

4.1 PASCAL 中的布尔（逻辑）类型

4.2 关系表达式与布尔表达式

4.3if 语句

4.4case 语句

4.1 PASCAL 中的布尔（逻辑）类型

在前面，我们学习了整型(integer)与实型（real）。其中 integer 型数据取值范围为-32768 到 32767 之间所有整数。而 real 型数据取值范围为 2.9e-39..1.7e38 之间的所有实数。它们都是数值型的（即值都为数）。布尔型（Boolean）是一种数据的类型，这种类型只有两种值，即“真”与“假”。

1、布尔常量

在 Pascal 语言中“真”用 ture 表示，“假”用 False 表示。所以布尔类型只有 true 与 false 两个常量。

2、布尔变量（boolean）

如果我们将某些变量说明成布尔型，那么这些变量就是布尔变量，它们只能用于存放布尔值（ture 或 false）。

例如，VAR A，B：BOOLEAN；

3、 布尔类型是顺序类型

由于这种类型只有两个常量，Pascal 语言中规定 ture 的序号为 1，false 的序号为 0。若某种类型的常量是有限的，那么这种类型的常量通常都有一个序号，我们称这种类型为顺序类型。如前面我们学过的整型（integer），以及后面要学到的字符型（char）都是顺序类型。

4、布尔类型的输入与输出

⑴输出

```
var a，b：boolean;
begin
  a：=true；b：=false;
  writeln（a，b）；
end.
```

输出为：truefalse

⑵布尔类型变量不能直接用读语句输入

布尔类型变量不能通过读语句给它们提供值。事实上，我们可以通过间接方式对布尔变量进行值的输入。

例如，以下程序是错误的：

```
var a,b,c:Boolean;
begin
  readln(a,b,c)；{错误语句}
  writeln(a,b,c);
end.
```

4.2 关系表达式与布尔表达式

1、什么是关系表达式

用小括号、>、<、>=、<=、=、<>将两个算术表达式连接起来的式子就称为关系表达式（比较式）。如：3+7>8,x+y<10,2*7<=13 等都是关系表达式。

2、关系表达式的值

很显然，这几个关系表达式中第一个是正确的，第三个是错误的，而第二个表达式可能是对的，也可能是错的。所以我们很容易发现，这些表达式的值是“对”的或“不对”的（或者说，是“真”的或“假”的），即关系表达式的值为布尔值。表示该比较式两端式子的大小关系是否成立。如 3+2>6 是错的，故它的值为 FALSE。同样，45>=32 是对的,故该表达式的值为 true。

关系表达式用于表示一个命题。如：“m为偶数”可表示为：m mod 2=0。“n为正数”可表示为：n>0。

3、布尔运算及布尔表达式

为了表示更复杂的命题，Pascal 还引入三种逻辑运算符：not、and、or。它们分别相当于数学上的“非”、“且”和“或”的意义。

运算符	运算	运算对象	结果类型
not	逻辑非	布尔型	布尔型
and	逻辑与	布尔型	布尔型
or	逻辑或	布尔型	布尔型
xor	逻辑异或	布尔型	布尔型

这三个运算符的运算对象为布尔量，其中 not 为单目运算，只有一个运算对象，and 与 or 为双目运算，有两个运算对象。它们的运算真值表如下：

a	b	Not a	a and b	a or b	a xor b
false	false	true	false	false	false
false	true	true	false	ture	true
true	false	false	false	true	true
true	true	false	true	true	false

于是，对于一个关系表达式，或多个关系表达式用布尔运算符连接起来的式子就称为布尔表达式。布尔表达式的值也为布尔值。

如果一个表达式里出现两个或两个以上的运算符，则必须规定它们的运算次序。pascal 规定：

⑴表达式中相同优先级的运算符，按从左到右顺序计算；

⑵表达式中不同优先级的运算符，按从高到低顺序计算；

⑶括号优先级最高，从内到外逐层降低。

对于一个复杂的表达式可能同时包含算术运算、关系运算和逻辑运算以及函数运算。运算的优先顺序为：括号、函数、not、*、/、div、mod、and、+、－、or、xor、关系运算。

对于复杂的命题，我们可以用布尔表达式来表示。例如，命题：“m,n 都是偶数或都是奇数”可表示为“(m mod 2=0)and(n mod 2=0) or (m mod 2=1)and(n mod 2=1)”。

4.3 if 语句

IF 语句是由一个布尔表达式和两个供选择的操作序列组成。运行时根据布尔表达式求值结果，选取其中之一的操作序列执行。有两种形式的 IF 语句：

格式一（称为“IF-THEN”型）：if 〈布尔表达式〉 then〈语句〉;

格式二（称为“IF-THEN-ELSE”型）：if 〈布尔表达式〉 then〈语句 1〉

else〈语句 2〉;

当布尔表达式的值为真，则执行 then 后面的语句，值为假时有两种情况：要么什么也不做，要么执行 else 后面的语句。注意 else 前面没有分号，因为分号是两个语句之间的分隔符，而 else 并非语句。如果在该处添了分号，则在编译的时候就会认为 if 语句到此结束，而把 else 当作另一句的开头，输出出错信息。

例如：求 y=f（x），当 x>0 时，y=1，当 x=0 时，y=0，当 x<0 时，y=-1。

程序如下：

```
program lianxi;
     var x,y:real;
   begin
     if x>0 then y:=1;
     if x=0 then y:=0;
     if x<0 then y:=-1;
     writeln('y=',y);
   end.
```

在 Turbo Pascal 语言 if 语句中被构造的语句只能是一条语句，当条件选择某个分支的计算要用多个语句描述时，就必须把该分支用 begin 和 end 括来，写成复合语句。在用 if 语句连续嵌套时，如果你插入适量的复合语句，有利于程序的阅读和理解。

例如：当 x>0 时候，计算 x*x，并且输出 x 和 x*x。

程序如下：

```
program lianxie3;
 var x,x1:real;
begin
 readln('x=',x);
 if x>0 then
   begin
     x1:=x*x;
     writeln('x*x=',x1);
     writeln('x=',x);
   end;
end.
```

例如：读入三个整数，找出并输出其中最大数。

程序如下：

```
program exec1(input,output);
  var n1,n2,n3,max:real;
begin
  read(n1,n2,n3);
  if n1>n2 then max:=n1
  else max:=n2;
    if n3>max then max:=n3;
    writeln('the largest number is ',max:8)
end.
```

例如：给出三角形的三个边长 A、B、C，求三角形面积。

程序如下：

```
program exec2(input,output);
    var a,b,c,s,area:real;
begin readln(a,b,c);
if (a+b>c)and(b+c>a)and(c+a>b)
then begin
  s:=(a+b+c)/2;
  area:=sqrt(s*(s-a)*(s-b)*(s-c));
  writeln('area=',area:6:2);
  end
else
begin
  writeln(a:6:2,b:6:2,c:6:2);
  writeln('input data error');
end;
end.
```

当 if 语句嵌套时，Turbo Pascal 约定 else 总是和最近的一个 if 配对。

在 if 语句中，如果 then 或 else 后的语句中又含有 IF 语句，则就构成 IF 语句嵌套结构。如果 IF 的格式 2 中 then 之后的语句仍为格式 2 形式的 IF 语句，那么应按缩进规则书写。例如：

```
program qt;
  var a,b,c,d,x:integer;
begin
  write('a=');read(a);
  write('b=');read(b);
  write('c=');read(c);
  write('d=');read(d);
  if a>b
    then if c>d
    then x:=1
    else x:=2
    else x:=3;
  writeln('x=',x);
  readln;
  readln;
end.
```

其语义和功能正确，表示 a>b 且 c>d 时，x 赋值 1；a>b 且 c≤d 时，x 赋值 2，而 a≤b 时，x 赋值 3。

如果 IF 语句的格式中 THEN 之后的语句仍为格式 1 形式的 IF 语句，那么这两个 IF 的条件可以合并为一个条件表达式，将 IF 的嵌套结构，化为一个 IF 语句。例如：

```
if a>b then if c>d then x:=1
```

表示 a>b 且 c>d 时，x 赋值 1，可以简化为：

```
if(a>b)and(c>d) then x:=1
```

两种写法作用相同，但简化性和运行时间后者好一些，对于多重嵌套的 IF 语句该问题更为突出，逻辑复杂。因此应尽量少用 IF 嵌套结构。

例如：计算 Y 的值，当 X<0 时，Y =2*X；当 X>0 时， Y =X/2 ；当 X=0 时，Y=0 (IF 语句的嵌套)。

```
program exec3(input,output);
  var x,y:real;
begin
  readln(x);
  if x>0
  then y:=x/2
  else
```

```
  if x<0
  then y:=2*x
  else y:=0;
  writeln('y=',y);
end.
```

4.4 case 语句

上面我们知道可以用嵌套的 if 语句实现多分支的选择结构。但是如果分支越来越多时，用嵌套的 if 语句实现多分支就显得繁杂。当多分支选择的各个条件由同一个表达式的不同结果值决定时，可以用 case 语句实现。它的选择过程，很象一个多路开关，即由 case 语句的选择表达式的值，决定切换至哪一语句去工作。因此在分支结构程序设计中，它是一种强有力的手段。在实现多路径分支控制时，用 case 对某些问题的处理和设计，比用 if 语句写程序具有更简洁、清晰之感。

㈠、语句的一般形式：

```
case <表达式> of
  <情况标号表 1>：语句 1;
  <情况标号表 2>：语句 2;
  ……
  <情况标号表 n>：语句 n
else  语句 n+1 {可选项}
end;
```

其中 case、of、end 是 Pascal 的保留字， 表达式的值必须是顺序类型，它可以是整型、布尔型及以后学习的字符型、枚举型和子界型。情况标号表是一串用逗号隔开的与表达式类型一致的常量序列。语句可以是任何语句，包括复合语句和空语句。

㈡、case 语句的执行过程

先计算表达式（称为情况表达式）的值，如果它的值等于某一个常量（称为情况常量，也称情况标号），则执行该情况常量后面的语句，在执行完语句后，跳到 case 语句的末尾 end 处。

㈢、说明

①情况表达式必须是顺序类型的；

②情况常量是情况表达式可能具有的值，因而应与情况表达式具有相同的类型；

③情况常量出现的次序可以是任意的；

④同一情况常量不能在同一个 case 语句中出现两次或两次以上；

⑤每个分语句前可以有一个或若干个用逗号隔开的情况常量；

⑥如果情况表达式的值不落在情况常的范围内，则认为本 case 语句无效，执行 case 语句的下一个语句。turbo pascal 中增加了一个“否则”的情况，即增加一个 else 子句，但也是可省的。

⑦每个常量后面只能是一个语句或一个复合语句。

例如：根据学生的成绩给予相应的等低，对应关系如下：

```
90--100    A
80--89     B
60--79     C
60 以下      D
program chengji;
 var s:integer;ch:char;
begin
 write('input the score: ');
 readln(s);
 if(s>=0)and(s<=100)then
   case s div 10 of
     10,9:ch:='B';
     8:ch:='B';
     7,6:ch:='C';
   else ch:='D';
 end;
writeln(s,'--',ch);
end.
```

例如：输入年、月，输出该月有几天。

```
program exec(input,output);
  var year,month,days:integer;
begin read(year,month);
  case month of
    1,3,5,7,8,10,12:days:=31;
    4,6,9,11:days:=30;
    2:if (year mod 4=0)and(year mod 100<>0)or(year mod 400=0)
      then days:=29
  else days:=28;
  end;
  writeln(year,month,days);
end.
```

例如：若已知 x 在 1 到 5 之间，按下公式计算 y 值。

$$Y\begin{cases}=3*x+10 & \text{当 } 1<=x<2\\ =2*\cos(x) & \text{当 } 2<=x<3\\ =\sin(x)/3 & \text{当 } 3<=x<5\end{cases}$$

```
program exec2(input,output);
  var x,y:real;
begin read(x);
  case trunc(x) of {trunc(x)是取整函数}
    1: y:=3*x+10;
    2: y:=2*cos(x);
    3,4: y:=sin(x)/3;
  end;
  writeln('y=',y);
end.
```

练习：

1、根据 x 的值，求函数 Y 的值：

$$y=\begin{cases}x+1, & 0<x<100\\ x-1, & 100\leqslant x\leqslant 200\\ -1, & \text{其它}\end{cases}$$

分析：利用 case 语句进行程序设计，关键在于巧妙地构造情况表达式。本例中三种情况可用一个表达式区分出来：Trunc(x/100)。因为 x 在(0～100)之间时表达式值为 0；x 在[100,200)时表达式值为 1 ；其余部分可用 else 子句表示。

源程序如下:

```
program ex;
var x,y:real;
begin
  write('Input x:');readln(x);
  case trunc(x/100) of
    0:y:=x+1;
    1:y:=x-1;
    else y:=0;
  end;{end of case}
  writeln('x=',x:8:2),'y=',y:8:2);
end.
```

2、输入一个年号,判断它是否是闰年。

分析：判断闰年的算法是：如果此年号能被 400 除尽， 或者它能被 4 整除而不能被 100 整除，则它是闰年。否则，它是平年。

源程序如下:

```
program ex;
var year:integer;
begin
  write('Input year:');readln(year);
  write(year:6);
  if (year mod 400=0 )  then
    writeln('is a leap year.')
  else
    if (year mod 4=0)and(year mod 100<>0)
    then writeln('is a leap year.')
    else writeln('is not a leap year.');
end.
```

3、判断 1995 年,每个月份的天数。

分析：程序分为:输入月份,计算该月的天数,输出天数

源程序如下：

```
program days;
var month,days:integer;
begin
write('Input month:');readln(month);
case month of
    1,3,5,7,8,10,12:days:=31;
    4,6,9,11 :days:=30;
    2 :days:=28;
    else days:=0;
  end;
  if days<>0 then writeln('Days=',days);
end.
```

4、期未来临了，班长小 Q 决定将剩余班费 X 元钱，用于购买若干支钢笔奖励给一些学习好、表现好的同学。已知商店里有三种钢笔，它们的单价为 6 元、5 元和 4 元。小 Q 想买尽量多的笔（鼓励尽量多的同学），同时他又不想有剩余钱。请您编一程序，帮小 Q 制订出一种买笔的方案。

分析：对于以上的实际问题，要买尽量多的笔，易知都买 4 元的笔肯定可以买最多支笔。因此最多可买的笔为 x div 4 支。由于小 q 要把钱用完，故我们可以按以下方法将钱用完：

若买完 x div 4 支 4 元钱的笔，还剩 1 元，则 4 元钱的笔少买 1 支，换成一支 5 元笔即可；若买完 x div 4 支 4 元钱的笔，还剩 2 元，则 4 元钱的笔少买 1 支，换成一支 6 元笔即可；若买完 x div 4 支 4 元钱的笔，还剩 3 元，则 4 元钱的笔少买 2 支，换成一支 5 元笔和一支 6 元笔即可。

从以上对买笔方案的调整，可以看出笔的数目都是 x div 4，因此该方案的确为最优方案。

源程序如下：

```
program pen;
  var a,b,c:integer;{a,b,c 分别表示在买笔方案中，6 元、5 元和 4 元钱笔的数目}
      x,y:integer;{x，y 分别表示剩余班费和买完最多的 4 元笔后剩的钱}
begin
  write('x=');readln(x){输入 x}
  c:=x div 4;{4 元笔最多买的数目}
  y:=x mod 4; {求买完 c 支 4 元笔后剩余的钱数 y}
  case y of
    0 : begin a:=0;b:=0; end;
    1 : begin a:=0;b:=1;c:=c-1; end;
    2 : begin a:=1;b:=0; c:=c-1;end;
    3 : begin a:=1;b:=1; c:=c-2;end;
  end;
```

```
  writeln('a=',a,'b=',b,'c=',c);
  end.
```

5、输入三角形的三个边,判断它是何类型的三角形(等边三角形？等腰三角形？一般三角形？)。

6、输入三个数，按由大到小顺序打印出来。

7、计算 1901 年 2099 年之间的某月某日是星期几。

8、输入两个正整数 a，b。b 最大不超过三位数，a 不大于 31。使 a 在左，b 在右，拼接成一个新的数 c。例如：a＝2，b＝16，则 c＝216；若 a＝18，b＝476，则 c＝18476。

提示：求 c 的公式为：

$$c = a \times K + b$$

其中：

$$K = \begin{cases} 10 & \text{当 B 为一位数时}(0<b<10) \\ 100 & \text{当 B 为二位数时}(10\leqslant b<100) \\ 1000 & \text{当 B 为三位数时}(100\leqslant b<1000) \end{cases}$$

第四章 循环结构程序设计

在 Pascal 语言中，循环结构程序通常由三种的循环语句来实现。它们分别为 FOR 循环、当循环和直到循环。通常将一组重复执行的语句称为循环体，而控制重复执行或终止执行由重复终止条件决定。因此，重复语句是由循环体及重复终止条件两部分组成。

4.1 while 语句 回顶

对于 for 循环有时也称为计数循环，当循环次数未知，只能根据某一条件来决定是否进行循环时，用 while 语句或 repeat 语句实现循环要更方便。

while 语句的形式为：

while <布尔表达式> do <语句>;

其意义为：当布尔表达式的值为 true 时，执行 do 后面的语句。

while 语句的执行过程为：

①判断布尔表达式的值,如果其值为真,执行步骤 2,否则执行步骤 4;

②执行循环体语句(do 后面的语句);

③返回步骤 1;

④结束循环,执行 while 的下一个语句。

说明：这里 while 和 do 为保留字，while 语句的特点是先判断，后执行。 当布尔表达式成立时，重复执行 do 后面的语句(循环体)。

while 语句用于“当满足某一条件时进行循环”的情况，也就是说 while 循环是是属于当型循环。为了能使 while 重复能终止，循环体中一定要有影响布尔表达式的操作，否则该循就是一个死循环。

保留关键字 DO 后面的“语句”只能是一条语句，称为“循环体”；如果循环体中需要包含多个语句则应该采用复合语句。

例如：计算从 0 到某个数之间所有奇数的和。

```
program jishu;
  var odds,limit,sum:integer;
begin
  readln(limit);
  sum:=0;
  odds:=1;
  while odds<=limit do
    begin
```

```
      sum:=sum+odds;
      odds:=odds+2
    end;
  writeln(sum:1)
end.
```

例如：

```
k:=10;
while k>0 do
begin
writeln(k);
k:=k-1
end
```

练习：

1 、求恰好使 s=1+1/2+1/3+…+1/n 的值大于 10 时 n 的值。

分析：“恰好使 s 的值大于 10”意思是当表达式 s 的前 n-1 项的和小于或等于 10，而加上了第 n 项后 s 的值大于 10。从数学角度，我们很难计算这个 n 的值。故从第一项开始，当 s 的值小于或等于 10 时，就继续将下一项值累加起来。当 s 的值超过 10 时，最后一项的项数即为要求的 n。

程序如下：

```
var
    s : real;
    n : integer; {n 表示项数}
begin
  s:=0.0;
  n:=0;
  while s<=10 do {当 s 的值还未超过 10 时}
    begin
      n:=n+1; {项数加 1}
      s:=s+1/n; {将下一项值累加到 s}
    end;
  writlen('n=',n); {输出结果}
end.
```

2、求两个正整数 m 和 n 的最大公约数。

分析：求两个正整数的最大公约数采用的辗转相除法求解。以下是辗转的算法：

分别用 m,n,r 表示被除数、除数、余数。

①求 m/n 的余数 r.

②若 r=0,则 n 为最大公约数.若 r≠0,执行第③步.

③将 n 的值放在 m 中,将 r 的值放在 n 中.

④返回重新执行第①步。

程序如下:

```
program ex4_4;
var m,n,a,b,r:integer;
begin
    write('Input m,n:');
    readln(m,n);
    a:=m;b:=n;r:=a mod b;
    while r<>0 do
    begin
      a:=b;b:=r;
      r:=a mod b;
    end;
    writeln('The greatest common divide is:',b:8);
end.
```

4.2 repeat-until 语句（直到循环） 回顶

repeat 语句用于"重复执行循环体，一直到指定的条件为真时为止"。语法格式为：

```
repeat
  语句 1;
 ……
  语句 n;
until  布尔表达式;
```

其中 Repeat、until 是 Pascal 保留字，repeat 与 until 之间的所有语句称为循环体。

说明：

①repeat 语句的特点是：先执行循环，后判断结束条件，因而至少要执行一次循环体。

②repeat-until 是一个整体，它是一个（构造型）语句，不要误认为 repeat 是一个语句，until 是另一个语句。

③repeat 语句在布尔表达式的值为真时不再执行循环体，且循环体可以是若干个语句，不需用 begin 和 end 把它们包起来， repeat 和 until 已经起了 begin 和 end 的作用。while 循环和 repeat 循环是可以相互转化的。

repeat 重复基本上有和 while 重复一样的描述循环计算的能力，但有一些不同：在 repeat 语句的结构中，布尔表达式求值在计算操作之后，而 while 语句中，布尔表达式求值在计算操作之前，也就是说 repeat 至少执行一次循环体。while 语句的成分语句只能是一个语句。因此，当重复动作包含多个语句时，要用 begin 和 end ，使它变成一个复合语句。而 repeat 语句的保留字 repeat 和 until 已经起语句括号作用，可以包含多个语句而无须 begin 和 end。repeat 语句中，当布尔表达式为 true 时结束循环，而 while 语句中，是当表达式为 false 时才结束循环。当描述由计算操作后的情况确定重复是否继续进行的计算时，通常用 repeat 语句描述。

对于上面练习 2 中求两个正整数的最大公约数，程序可用 repeat-until 循环实现如下：

```
var
    m,n,a,b,r : integer;
begin
  write('Input m,n=');
  readln(m,n);
  a:=m;b:=n;
  repeat
    r:=a mod b;
    a:=b;b:=r;
  until r=0;
  writeln('The greatest common divide is',a);
end.
```

例如：以下循环求 sum=1+2+3+.....+100 的和。

```
program he;
var n,q,sum:integer;
begin
write('q=');
readln(q);
sum:=0;
n:=1;
repeat
sum:=sum+n;
n:=n+1;
until n>q;
writeln ('sum=',sum);
readln
end.
```

4.3 for 语句 回顶

for 语句用来描述已知重复次数的循环结构。for 语句有两种形式：

(1) for 控制变量：＝初值 to 终值 do 语句；

(2) for 控制变量：＝初值 downto 终值 do 语句；

第一种形式的 for 语句是递增循环。首先将初值赋给控制变量，接着判断控制变量的值是否小于或等于终值，若是，则执行循环体，在执行了循环体之后，自动将控制变量的值该为它的后继值，并重新判断是否小于或等于终值。当控制变量的值大于终值时，退出 for 循环，执行 for 语句之后的语句。

第二种形式的 for 语句是递减循环。首先将初值赋给控制变量，接着判断控制变量的值是否大于或等于终值，若是，则执行循环体，在执行了循环体之后，自动将控制变量的值该为它的前趋值，并重新判断是否大于或等于终值。当控制变量的值小于终值时，退出 for 循环，执行 for 语句之后的语句。

说明：

①循环控制变量必须是顺序类型。例如，可以是整型、字符型等，但不能为实型。

②循环控制变量的值递增或递减的规律是：选用 to 则为递增；选用 downto 则递减。

③所谓循环控制变量的值"超过"终值，对递增型循环，"超过"指大于，对递减型循环，"超过"指小于。

④循环体可以是一个基本语句，也可以是一个复合语句。

⑤循环控制变量的初值和终值一经确定，循环次数就确定了。但是在循环体内对循环变量的值进行修改，常常会使得循环提前结束或进入死环。建议不要在循环体中随意修改控制变量的值。

⑥for 语句中的初值、终值都可以是顺序类型的常量、变量、表达式。

例：for i:=5 to 10 do writeln (i);

输出的结果为： 5 6 7 8 9 10 即循环一共执行了 6 次。

例：计算 1+2+3+……+99+100 的和。

```
program jia;
var i,n,sum:integer;
begin
sum:=0;
for i:=1 to 100 do
sum:=sum+i;
writeln(sum);
readln
end.
```

如果要重复多个语句，一定要用 begin-end 形式：

例：

```
program hb;
var i:integer;
begin
for i:=1 to 10 do
begin
writeln('i=',i);
writeln('h=',10-i);
end;
readln;
end.
```

FOR 循环的几点注意内容：

（1）循环控制变量必须是顺序类型的变量。所谓顺序类型的变量，就是指整型，字符型，枚举型，子界型，不允许是实型。

（2）不允许在循环体内再对循环控制变量赋值。

例如：

```
a:=10;b:=50;
for k:=a to b do
begin
k:=k+1;{这一句是错误的！！！！！！！！！}
writeln (k);
end;
```

（3）当循环初值或循环终值中包含变量时，允许在循环体内改变这些变量的值，并不改变原定的循环次数。

例：

```
a:=1;b:=10;
for i:=a to b do
begin
a:=5;b:=4;
end;
```

在上面例子中，A，B 的值在循环的内部发生了变化，但并不影响循环的次数，依然是 10 次。

（4）多重循环：循环体由 PASCAL 语句构成，当然也可以包含 FOR 语句，这就构成了循环的嵌套，形成多重循环。

例如，以下 FOR 循环输出 5 行，每行输出 10 个星号(*)

```
for i:=1 to 5 do
begin
for j:=1 to 10 do
write('*');
end;
```

初学者应当特别注意，内层的循环变量不能和外层的循环变量相同。也就是说，嵌套的各层循环应当使用不同的变量作为循环变量。

以上我们已介绍了三种循环语句。一般说来，用 for 循环比较简明，只要能用 for 循环，就尽量作用 for 循环。只在无法使用 for 循环时才用 while 循环和 repeat-until 循环， 而且 while 循环和 repeat-until 循环是可以互相替代的。for 循环在大多数场合也能用 whiel 和 repeat-until 循环来代替。一般 for 循环用于有确定次数循环，而 while 和 repeat-until 循环用于未确定循环次数的循环。

当一个循环的循环体中又包含循环结构程序时，我们就称之为循环嵌套。

例 1. 输出 1－100 之间的所有偶数。

```
var i:integer;
begin
  for i:=1 to 100 do
  if i mod 2=0 then write(i:5);
end.
```

例 2. 求 N! ＝1＊2＊3＊…＊N ，这里 N 不大于 10。

分析：程序要先输入 N，然后从 1 累乘到 N。

程序如下：

```
var
    n,i : integer; {i 为循环变量}
    S : longint;{s 作为累乘器}
begin
  write('Enter n=');readln(n);{输入 n}
s:=1;
  for i:=2 to n do{从 2 到 n 累乘到 s 中}
    s:=s*i;
  writeln(n,'!=',s);{输出 n!的值}
end.
```

练　习

1.　求 s=1+4+7+…+298 的值。

2.　编写一个评分程序，接受用户输入 10 个选手的得分(0-10 分)，然后去掉一个最高分和一个最低分，求出某选手的最后得分(平均分)。

3.　用一张一元票换 1 分、2 分和 5 分的硬币，每种至少一枚， 问有哪几种换法（各几枚）？

4.4　goto 语句和标号 回顶

goto 语句是一种无条件转向语句，它可以控制直接从程序的一条语句转向另一条语句。goto 语句的语法形式为：

```
        goto  标号;
```

其中标号必须是不超过 4 位整数的正整数或标识符组成，但标号必须在说明语句中先予以说明。

goto 语句会使程序出现一种称为“乱面条”的结构，因此你最好还是不要去用。

例:编写计算 y=lnx,输入一个实数 x,若 x>0,则输出计算结果,若 x<=0,则显示无解标志“cannot solve”。

```
Program hb(input,output);
  Label a10;
  Var x,y:real;
Begin
  Write('input real data:');
  Readln(x);
  If x<=0
    Then begin writeln('cannot solve');
    goto a10;
  end;
  Y:=ln(x);
  Writeln('y=',y:6:2);{输出指定宽度的结果}
  A10: {标号后为空语句，表示空操作}
End.
```

4.5 循环结构程序设计 回顶

例 1：求 1!+2!+…+10!的值。

分析：这个问题是求 10 自然数的阶乘之和，可以用 for 循环来实现。程序结构如下：

```
for n:=1 to 10 do
begin
  ①N!的值 t
  ②累加 N!的值 t
end
```

显然,通过 10 次的循环可求出 1!,2!…,10!,并同时累加起来, 可求得 S 的值。而求 T=N!,又可以用一个 for 循环来实现:

```
t=1;
for j:=1 to n do
  t:=t*j;
```

因此,整个程序为:

```
program ex4_5;
var t,s:real;
   i,j,n:integer;
begin
  S:=0;
  for n:=1 to 10 do
  begin
    t:=1;
    for j:=1 to n do
      t:=t*j;
    S:=S+t;
  end;
  writeln('s=',s:0:0);
end.
```

以上的程序是一个二重的 for 循环嵌套。这是比较好想的方法，但实际上对于求 n!，我们可以根据求出的(n-1)!乘上 n 即可得到，而无需重新从 1 再累乘到 n。

程序可改为：

```
program ex4_5;
var t,s:real;
```

```
    i,j,n:integer;
begin
  S:=0;t:=1;
  for n:=1 to 10 do
  begin
    t:=t*n;
    S:=S+t;
  end;
  writeln('s=',s:0:0);
end.
```

显然第二个程序的效率要比第一个高得多。第一程序要进行 1+2+…+10=55 次循环，而第二程序进行 10 次循环。如题目中求的是 1！＋2！＋…＋1000！，则两个程序的效率区别更明显。

例 2： 一个炊事员上街采购，用 500 元钱买了 90 只鸡，其中母鸡一只 15 元，公鸡一只 10 元，小鸡一只 5 元，正好把钱买完。问母鸡、公鸡、小鸡各买多少只？

分析：设母鸡 I 只，公鸡 J 只，则小鸡为 90-I-J 只，则 15*I+ 10* J+(90-I-J)*5=500，显然一个方程求两个未知数是不能直接求解。必须组合出所有可能的 I，j 值，看是否满足条件。这里 I 的值可以是 0 到 33，J 的值可以 0 到 50。

源程序如下：

```
programr ex4_6;
var i,j,k:integer;
begin
  for i:=1 to 5 do
  for j:=1 to 8 do
    begin
      k:=90-i-j;
      if 15*i+10*j+5*k=500 then writeln(i:5,j:5,k:5);
    end;
end.
```

例 3：求 100－200 之间的所有素数。

分析：我们可对 100－200 之间的每一整数进行判断，判断它是否为素数，是则输出。而对于任意整数 i，根据素数定义，我们从 2 开始，到 ，找 i 的第一个约数。若找到第一个约数，则 i 必然不是素数。否则 i 为素数。

源程序如下：

```
var
    i : integer;
    x : integer;
begin
  for i:=100 to 200 do
    begin
      x:=2;
      while (x<=trunc(sqrt(i)))and(i mod x<>0)do
        begin
          x:=x+1;
        end;
      if x>trunc(sqrt(i)) then write(i:8);
    end;
end.
```

练习：

1、输入一个正整数 n，将 n 分解成质因数幂的乘积形式。

例如：36=22*32

2、输出如下图形。（三个并列，只画出第一个三角形）

```
  *
 ***
```

*****
****
****

3、编写一程序，验证角谷猜想。所谓的角谷猜想是：″对于任意大于 1 的自然数 n，若 n 为奇数，则将 n 变为 3*n+1，否则将 n 变为 n 的一半。经过若干次这样的变换，一定会使 n 变为 1。″

4．有一堆 100 多个的零件，若三个三个数，剩二个；若五个五个数，剩三个；若七个七个数，剩五个。请你编一个程序计算出这堆零件至少是多少个？

第五章 枚举型和子界型

.1 类型定义

类型定义的语法格式：

```
type
 <标识符 1>=<类型 1>;
 <标识符 1>=<类型 1>;
 ……
 <标识符 n>=<类型 n>;
```

在前面几章中我们用到了整型、实型、布尔型、字符型的数据。以上数据类型是由 pascal 规定的标准数据类型，只要写 integer,real,boolean, char, pascal 编译系统就能识别并按这些类型来处理。pascal 还允许用户定义一些类型，这是其它一些语言所没有的，这就使得 pascal 使用灵活、功能丰富。

.2 枚举类型

随着计算机的不断普及，程序不仅只用于数值计算，还更广泛地用于处理非数值的数据。例如，性别、月份、星期几、颜色、单位名、学历、职业等，都不是数值数据。

在其它程序设计语言中，一般用一个数值来代表某一状态，这种处理方法不直观，易读性差。如果能在程序中用自然语言中有相应含义的单词来代表某一状态，则程序就很容易阅读和理解。也就是说，事先考虑到某一变量可能取的值，尽量用自然语言中含义清楚的单词来表示它的每一个值，这种方法称为枚举方法，用这种方法定义的类型称枚举类型。

枚举类型是一种很有实用价值的数据类型，它是 pascal 一项重要创新。

.21 枚举类型的定义

用整数描述非数值型数据不准确，无法反映它们的特征，于是使用自定义方法，直接用符号表示，例如星期一用 MON 表示，一月用 JAN 表示等等。这种将非数值数据一一列出的数据类型，称为枚举类型。

通过预定义列出所有值的标识符来定义一个有序集合，这些值的次序和枚举类型说明中的标识符的次序是一致的。

枚举类型的形式：type 类型名=(标识符 1，……，标识符 n)

例如星期几、六种颜色、四种文具的枚举类型定义：

```
type weekday=(sun,mon,tue,wed,thu,fri,sat);
     color=(red.orange,yellow,green,blue,purple);
     sup=(pen,pencil,ink,buler);
```

枚举元素只能是标识符，而不能是数值常量或字符常量。例如以下的定义是错误的：

type daystype=('sun','mon','tue','wed','thu','fri','sat')

枚举元素是标识符，不要把作为枚举元素的标识符视作变量名，它不能被赋值。同一个枚举元素不能出现在两个或两个以上的枚举类型定义中。例如以下的定义是错误的：

type daytype1=(monday,tuesday);

daytype2=(monday,wednesday);

类型定义之后，即可在变量说明中使用，例如：

var day:weekday;c:color;s:sup;

以上是类型定义和变量说明是分开进行的，可以将枚举类型的定义和变量的定义结合在一起。

例如：var day: (sun,mon,tue,wed,thu,fri,sat);

c: (red.orange,yellow,green,blue,purple);

s: (pen,pencil,ink,buler);

.22 枚举类型的性质

1、枚举类型属于顺序类型。根据定义类型时各枚举元素的排列顺序确定它们的序列，序列号从0开始，以下递增1。枚举类型变量的取值范围只能是类型定义中括号内所列出一个个标识符。

例如：设有定义：

type days=(sun,mon,tue,wed,thu,fri,sat);

则：（ord取序号，succ后继，pred前导）

ord(sun)=0,ord(mon)=1,ord(sat)=6;succ(sun)=mon,succ(mon)=tue,

succ(fri)=sat;pred(mon)=sun,pred(tue)=mon,pred(sat)=fri。

应注意的是：枚举类型中的第一个元素无前趋，最后一个元素无后继。

2、对枚举类型只能进行赋值运算和关系运算

一旦定义了枚举类型及这种类型的变量，则在语句部分只能对枚举类型变量赋值，或进行关系运算，不能进行算术运算和逻辑运算。

在枚举元素比较时，实际上是对其序号的比较。当然，赋值或比较时，应注意类型一致。

例如，设程序有如下说明：

type days=(sun,mon,tue,wed,thu,fri,sat);

colors=(red,yellow,blue,white,black,green);

var color:colors;

weekday:days;

则下列比较或语句是合法的:

weekday:=mon;

if weekday=sun then write('rest');

weekday<>sun

而下面比较或语句是不合法的:

mon:=weekday;

mon:=1;

if weekday=sun or sat then write('rest');

sun>red

weekday<>color

枚举类型是有序的，可进行关系运算，其大小按序号确定，比较结果为布尔值。例：

sun<mon 值为true red>blue 值为false false>true 值为false

3、枚举变量的值只能用赋值语句来获得

也就是说，不能用read(或readln)读一个枚举型的值。同样，也不能用write(或writeln)输出一个枚举型的值。如write(red)是非法的，会发生编译错误。千万不要误认为，该语句的结果是输出"red"三个字符。

但对枚举数据的输入与输出可通过间接方式进行。输入时，一般可输入一个代码，通过程序进行转换，输出时，也只是打印出与枚举元素相对应的字符串。这在后面的例题中将有使用示例。

5、同一个枚举元素不能出现在两个或两个以上的枚举类型定义中。

如:

type color1=(red,yellow,white);

color2=(blue,red,black);

是不允许的，因为red属于两个枚举类型。

.23 枚举类型的函数

以枚举类型数据为自变量的函数有三种：pred、succ、ord。

1、前导函数 pred

前导函数的结果是自变量的前驱值仍为枚举数据。例如：

pred(sat)值为 fri

即星期六的前驱是星期五。注意序号为 0 的枚举数据无前导值。

2、后继函数 succ

后继函数的结果是自变量的后继值，例如赋值运算的结果：

week:=succ(fri);week 的内容为 sat.

注意：序号最大的枚举数据的后继值是没有定义的。

3、序号函数 ord

此函数的自变量为枚举数据，函数值为非负整数。例如：

ord(sun)=0、ord(sat)=6、ord(true)=1

但是枚举类型中的第一个元素没有前趋，最后一个元素没有后继。

.24 枚举类型的输入输出

Turbo Pascal 不允许直接读写枚举值，所以枚举值的输出常用 case 语句间接的输出。枚举值的输入，则要一一判断读入字符是否是枚举类型的标识符。若是才能赋给枚举变量，否则就会出错。

例如，对于星期几的七个枚举数据的输入利用 CASE 语句的转换结构为：

```
case k of
  0:w:=sun;
  1:w:=mon;
  2:w:=tue;
  3:w:=wed
  4:w:=thu;
  5:w:=fri;
  6:w:=sat;
  end;
```

输出时，利用字符串的可输出性，通过 CASE 语句进行转换输出。对于上例是星期几的输出为：

```
case w of
 sun:write('sunday');
 mon:write('monday');
 tue:write('tuesday');
 wed:write('wednesday');
 thu:write('thursday');
 fri:write('friday');
 sat:write('saturday');
end;
```

.25 枚举类型应用举例

例如：一周七天用 sun,mon,tue,wed,thu,fri,sat 表示， 要求利用枚举类型编程：当输入星期几的数字,能输出它的后一天是星期几(也用英文表示)。

源程序如下：

```
program ex6_1;
type week=(sun,mon,tue,wed,thu,fri,sat);
var
    i : integer;
    day,sucday : week;
begin
  write('What date is it');readln(i);
  case i of {根据输入 i 转换成枚举型}
    1:day:=mon;
    2:day:=tue;
    3:day:=wed;
    4:day:=thu;
```

```
    5:day:=fri;
    6:day:=sat;
    7:day:=sun;
  end;
  {计算明天 sucday}
  if (day=sat) then sucday:=sun
  else sucday:=succ(day);
  {输出明天是星期几}
  write('The next day is ');
  case sucday of
    sun:writeln('sunday');
    mon:writeln('monday');
    tue:writeln('tuesday');
    wed:writeln('wednesay');
    thu:writeln('thursday');
    fri:writeln('friday');
    sat:writeln('saturday');
  end;
  end.
```

评注：程序中变量 day、sucday 分别表示今天、明天。

.3　子界类型

如果我们定义一个变量 i 为 integer 类型，那么 i 的值在微型机系统的 pascal 中,使用 2 字节的定义表示法,取值范围为-32768～32767。而事实上，每个程序中所用的变量的值都有一个确定的范围。

例如，人的年龄一般不超过 150，一个班级的学生不超过 100 人，一年中的月数不超过 12，一月中的天数不超过 31，等等。

. 31 子界类型定义

子界类型是由整型、字符型、枚举型、布尔型的两个常量指定该类型的值域区间。子界类型的形式：

type〈类型名〉〈常量 1〉. .〈常量 2〉

说明:

①其中类型名是由用户指定，是一个合法的标识符。常量 1 称为子界的下界,常量 2 称为子界的上界。

②下界和上界必须是同一顺序类型（该类型称为子界类型的基类型），且上界的序号必须大于下界的序号。

例如，下列说明：

```
type age=0.5..150;
   letter=0..'z';
   let1='z'..'a';
```

都是错误的。

③可以直接在变量说明中定义子界类型。如：

```
type letter='a'..'d';
var ch1,ch2:letter;
```

可以合并成:

```
var ch1,ch2:'a'..'d';
```

当然，将类型定义和变量定义分开，则更为清晰。

. 32 子界类型数据的运算规则

⑴凡可使用基类型的运算规则同样适用该类型的子界类型。

例如，可以使用整型变量的地方，也可以使用以整型为基类型的子界类型数据。

⑵对基类型的运算规则同样适用于该类型的子界类型。

例如，div，mod 要求参加运算的数据为整, 因而也可以为整型的任何子界类型数据。

⑶基类型相同的不同子界类型数据可以进行混合运算。

例如：设有如下说明：

```
type a=1..100;
 b=1..1000;
     c=1..500;
var
      x:a;
      y:b;
      t:c;
      z:integer;
```

则下列语句也是合法的：

```
Z:=Sqr(x)+y+t;
```

下列语句:

```
t:=x+y+z;
```

当 X+Y+Z 的值在 1～500 范围内时是合法的，否则会出错。

. 33 子界类型应用举例

例 1 利用子界类型作为情况语句标号，编一个对数字， 大小写字母和特殊字符进行判别的程序。

源程序如下：

```
program cas;
var c:char;
begin
  readln(c);
case c of
  '0'..'9':writeln('digits');
  'A'..'Z':writeln('UPPER-CASELETTERS');
  'a'..'z':writeln('lower-caseletters');
  esle writeln('special charactors');
end;
end.
```

例 2 使用子界型情况语句，当输入月、日、年(10 /30/1986)，输出 30 Oct 1986。

源程序如下:

```
program ex6_3;
  var month:1..12;
      date:1..31;
      year:1900..1999;
  begin
    write('Enter date(mm-dd-yy):');
    readln(month,date,year);
    write(date);
    case month of
      1:write('Jan':5);
      2:write('Feb':5);
      3:write('Mar':5);
      4:write('Apr':5);
      5:write('May':5);
      6:write('Jun':5);
      7:write('Jul':5);
      8:write('Aug':5);
      9:write('Sep':5);
      10:write('Oct':5);
      11:write('Nov':5);
      12:write('Dec':5);
    end;
    writeln(year:7);
```

```
  end.
```

枚举类型和子界类型均是顺序类型，在数组的定义时，实际上我们已经用到了子界类型，数组中的下标类型确切地讲可以是枚举类型或子界类型，大多数情况下用子界类型。

如有如下说明：

```
type color=(red,yellow,blue,white,black);
var
    a:array[color]of integer;
    b:array[1..100]of color;
```

都是允许的。

例 3　按月、日、年顺序读入一日期，输出该日期是这一年中的第几天。

```
program date;
  var year:0..2010;
      month,i:1..12;
      day:1..31;
      dayth:integer;
begin
  read(month,day,year);
  dyath:=0;
  for i:=1 to month-1 do
    case i of
      1,3,5,7,8,10,12:dayth:=dayth+31;
      2:if ((year mod 4=0)and(year mod 100<>0)or(year mod 400 =0)
        then dayth:=dayth+29
        else dayth=:=dayth+28;
      4,6,9,11:dayth:=dayth+30;
    end;
  dayth:=dayth+day;
  writeln(dayth)
end.
```

.4　类型相容和赋值相容

1.类型相容性

类型相容是对参加同一运算的两个对象的类型要求。设有两个变量，如果满足下列条件之一，就说这两个变量的类型相容。

(1)两个变量的类型相同

a.两个变量被同一类型说明。

例如：var a,b:1..30;

b.两个变量的类型是同一类型标识符。

例如：var a:1..30;
　　b:1..30;

c.两个变量的类型是不同的类型标识符，但在类型定义中已经说明两个标识符相同。

例如：type date=1..100;
　　range=date;
　　var m:data;
　　n:range;

(2)一个变量的类型是另一个变量的子界。

(3)两个变量的类型都是同一基类型的子界。

(4)两个变量的类型是基类型相容的集合类型。

(5)两个变量的类型是成分相同的串类型。

2.赋值相容性

赋值相容是对赋值操作的两个对象的类型要求。设赋值语句″：＝″左边的变量类型为 T，右边表达式的类型为 E,若类型 T 和类型 E 满足下列条件之一，则称他们是赋值相容的。

(1)T 和 E 是相同的类型，而且类型不是文件类型，也不是具有文件类成分的构造类型。

(2)T 是实型，而 E 是整型或整型的子界。

(3)T 和 E 是类型相容的顺序类型，并且 E 的值不超出 T 所定义的值的范围

(4)T 和 E 是类型相容的集合类型，并且 E 的值不超出 T 所定义的值的范围

(5)T 和 E 是类型相容的串类型。

当 T 和 E 是顺序类型或都是集合类型时，不仅要求这两个类型是相容的，而且要求 E 的值不超出 T 所定义的值的范围；否则将产生类型溢出，而这种错误只能在你运行程序时进行检查，因此你必须要避免不发生这种错误。

第六章 数组

.1 数组

.11 为什么要使用数组

例 1 输入 50 个学生的某门课程的成绩，打印出低于平均分的同学号数与成绩。

分析：在解决这个问题时，虽然可以通过读入一个数就累加一个数的办法来求学生的总分，进而求出平均分。但因为只有读入最后一个学生的分数以后才能求得平均分，且要打印出低于平均分的同学，故必须把 50 个学生的成绩都保留下来， 然后逐个和平均分比较，把高于平均分的成绩打印出来。如果，用简单变量 a1，a2,...，a50 存放这些数据，可想而知程序要很长且繁。

要想如数学中使用下标变量 ai 形式表示这 50 个数，则可以引入下标变量 a[i]。这样问题的程序可写为：

```
tot:=0;{tot 表示总分}
for i:=1 to 50 do {循环读入每一个学生的成绩，并累加它到总分}
  begin
    read(a[i]);
    tot:=tot+a[i];
  end;
ave:=tot/50;{计算平均分}
for i:=1 to 50 do
  if a[i]<ave then writeln('No.',i,' ',a[i]);{如果第 i 个同学成绩小于平均分，则将输出}
```

而要在程序中使用下标变量，则必须先说明这些下标变量的整体－数组，即数组是若干个同名（如上面的下标变量的名字都为 a）下标变量的集合。

7.12 一维数组

当数组中每个元素只带有一个下标时，我们称这样的数组为一维数组。

1、一维数组的定义

（1）类型定义

要使用数组类型等构造类型以及学习的自定义类型（枚举类型与子界类型），应在说明部分进行类型说明。 这样定义的数据类型适用整个程序。

类型定义一般格式为：

```
type
  <标识符 1>=<类型 1>;
  <标识符 2>=<类型 2>;
  ……
  <标识符 n>=<类型 n>;
```

其中 type 是 Pascal 保留字，表示开始一个类型定义段。在其后可以定义若干个数据类型定义。<标识符>是为定义的类型取的名字, 称它为类型标识符。

类型定义后，也就确定了该类型数据取值的范围，以及数据所能执行的运算。

（2）一维数组类型的定义

一维数组类型的一般格式：

```
type
    数组类型名= array[下标 1..下标 2] of <基类型>;
```

说明：其中 array 和 of 是 pascal 保留字。下标 1 和下标 2 是同一顺序类型，且下标 2 的序号大于下标 1 的序号。下标类型可以是布尔类型、字符类型及任何用户定义的枚举类型或子界类型，绝不可以是实数类型。它给出了数组中每个元素(下标变量) 允许使用的下标类型，也决定了数组中元素的个数。基类型是指数组元素的类型，它可以是任何类型，同一个数组中的元素具有相同类型。因此我们可以说，数组是由固定数量的相同类型的元素组成的。

再次提请注意：类型和变量是两个不同概念，不能混淆。就数组而言，程序的执行部分使用的不是数组类型（标识符）而是数组变量（标识符）。

一般在定义数组类型标识符后定义相应的数组变量，如：

```
type arraytype = array [1..8] of integer;
var a1,a2:arraytype;
```

其中 arraytype 为一个类型标识符，表示一个下标值可以是 1 到 8，数组元素类型为整型的一维数组；而 a1,a2 则是这种类型的数组变量。

也可以将其全并起来：

```
var a1,a2:array[1..8]of integer;
```

当在说明部分定义了一个数组变量之后，pascal 编译程序为所定义的数组在内存空间开辟一串连续的存储单元。

例如，设程序中有如下说明：

```
type rowtype=array[1..8]of integer;
     coltype=array['a'..'e']of integer;
var a:rowtype;b:coltype;
```

2、一维数组的引用

当定义了一个数组，则数组中的各个元素就共用一个数组名(即该数组变量名)，它们之间是通过下标不同以示区别的。 对数组的操作归根到底就是对数组元素的操作。一维数组元素的引用格式为：

数组名[下标表达式]

说明：

下标表达式值的类型，必须与数组类型定义中下标类型完全一致，并且不允许超越所定义的下标下界和上界。

数组是一个整体，数组名是一个整体的标识，要对数组进行操作，必须对其元素操作。数组元素可以象同类型的普通变量那样作用。如：a[3]:=34;是对数组 a 中第三个下标变量赋以 34 的值。read(a[4]);是从键盘读入一个数到数组 a 第 4 个元素中去。

特殊地，如果两个数组类型一致,它们之间可以整个数组元素进行传送。如：

```
var a,b,c:array[1..100] of integer;
begin
  c:=a;a:=b;b:=c;
end.
```

在上程序中，a,b,c 三个数组类型完全一致， 它们之间可以实现整数组传送，例子中，先将 a 数组所有元素的值依次传送给数组 c，同样 b 数组传给 a，数组 c 又传送给 b，上程序段实际上实现了 a,b 两个数组所有元素的交换。

对于一维数组的输入与输出, 都只能对其中的元素逐个的输入与输出。在下面的应用示例中将详细介绍。

3、一维数组应用示例

例 1：输入 50 个数，要求程序按输入时的逆序把这 50 个数打印出来。也就是说，请你按输入相反顺序打印这 50 个数。

分析：我们可定义一个数组 a 用以存放输入的 50 个数, 然后将数组 a 内容逆序输出。

源程序如下：

```
program ex5_1;
type arr=array[1..50]of integer; {说明一数组类型 arr}
var a:arr;i:integer;
begin
  writeln('Enter 50 integer:');
  for i:=1 to 50 do read(a[i]); {从键盘上输入 50 个整数}
  readln;
  for i:=50 downto 1 do {逆序输出这 50 个数}
    write(a[i]:10);
end.
```

例 2：输入十个正整数，把这十个数按由小到大的顺序排列。

将数据按一定顺序排列称为排序，排序的算法有很多，其中选择排序是一种较简单的方法。

分析：要把十个数按从小到大顺序排列，则排完后，第一个数最小，第二个数次小，……。因此，我们第一步可将第一个数与其后的各个数依次比较，若发现，比它小的，则与之交换，比较结束后，则第一个数已是最小的数（最小的泡往下冒）。同理，第二步，将第二个数与其后各个数再依次比较，又可得出次小的数。如此方法进行比较，最后一次，将第九个数与第十个数比较，以决定次大的数。于是十个数的顺序排列结束。

例如下面对5个进行排序，这个五个数分别为8　2　9　10　5。按选择排序方法，过程如下：

初始数据　：8　2　9　10　5
第一次排序：8　2　9　10　5
　　9　2　8　10　5
　　10　2　8　9　5
　　10　2　8　9　5
第二次排序：10　8　2　9　5
　　10　9　2　8　5
　　10　9　2　8　5
第三次排序：10　9　8　2　5
　　10　9　8　2　5
第四次排序：10　9　8　5　2

对于十个数，则排序要进行9次。源程序如下：

```
program ex5_2;
  var a:array[1..10]of integer;
        i,j,t:integer;
    begin
      writeln('Input 10 integers:');
      for i:=1 to 10 do read(a[i]);{读入10个初始数据}
      readln;
      for i:=1 to 9 do{进行9次排序}
        begin
          for j:=i+1 to 10 do{将第i个数与其后所有数比较}
              if a[i]<a[j] then {若有比a[i]大,则与之交换}
                begin
                  t:=a[i];a[i]:=a[j];a[j]:=t;
                end;
          write(a[i]:5);
        end;
    end.
```

例3：读入5个学生的学号和成绩，计算他们的平均分，若比平均分高10分的等第为A，若比平均分高小于10分的等地为B，若低于平均分，则等第为C，输出他们的成绩和等第。

```
program sample7d1(input,output);
  const n=5;
  type
    no=array[1..n] of integer;
    s=array[1..n]of real;
  var
    i:integer;
    k:real;
    num:no;
    score:s;
begin
  k:=0;
  for i:=1 to n do
    begin
      readln(num[i],score[i]);
      k:=k+score[i];
```

```
    end;
  k:=k/n;
  for i:=1 to n do
    begin
      write(num[i],score[i]);
      if (score[i]-k)>=10 then writeln('A')
        else if((score[i]-k)<10)and((score[i]-k)>0) then writeln('B')
          else writeln('C');
    end;
end.
```

练习：

1. 输入一串小写字母（以"."为结束标志），统计出每个字母在该字符串中出现的次数（若某字母不出现，则不要输出）。

例：

输入：aaaabbbccc.

输出：a:4
　　b:3
　　c:3

2. 输入一个不大于 32767 的正整数 N，将它转换成一个二进制数。

例如：

输入：100

输出： 1100100

3. 输入一个由 10 个整数组成的序列，其中序列中任意连续三个整数都互不相同，求该序列中所有递增或递减子序列的个数。

例如：

输入：1 10 8 5 9 3 2 6 7 4

输出：6

对应的递增或递减子序列为：

1 10
10 8 5
5 9
9 3 2
2 6 7
7 4

7.13 多维数组

1、多维数组的定义

当一维数组元素的类型也是一维数组时，便构成了二维数组。二维数组定义的一般格式：

type 数组类型名= array[下标类型 1] of array[下标类型 2] of 元素类型；

但我们一般这样定义二维数组：

type 数组类型名= array[下标类型 1，下标类型 2] of 元素类型；

说明：其中两个下标类型与一维数组定义一样，可以看成“下界 1..上界 1”和“下界 2..上界 2”，给出二维数组中每个元素(双下标变量)可以使用下标值的范围。of 后面的元素类型就是基类型。

二维数组的变量说明可以表示成：

```
type reals = array [1…m, 1…n] of real;
var date1, date2: reals;
```

也可以表示成：

```
var date1, date2: array[1…m, 1…n] of real;
```

一般地，n 维数组的格式为：

type 数组类型名= array[下标类型 1，下标类型 2，...，下标类型 n] of 元素类型；

其中，下标类型的个数即数组的维数，且说明了每个下标的类型及取值范围。

2、多维数组元素的引用

多维数组的数组元素引用与一维数组元素引用类似，区别在于多维数组元素的引用必须给出多个下标。

引用的格式为：

〈数组名〉[下标 1,下标 2,...,下标 n]

说明：显然，每个下标表达式的类型应与对应的下标类型一致，且取值不超出下标类型所指定的范围。

例如，设有说明：

```
type matrix=array[1..5,1..4]of integer;
var a:matrix;
```

则表示 a 是二维数组，共有 5*4=20 个元素，它们是：

a [1，1] a [1，2] a [1，3] a [1，4]
a [2，1] a [2，2] a [2，3] a [2，4]
a [3，1] a [3，2] a [3，3] a [3，4]
a [4，1] a [4，2] a [4，3] a [4，4]
a [5，1] a [5，2] a [5，3] a [5，4]

因此可以看成一个矩阵，a [4，2] 即表示第 4 行、第 2 列的元素。由于计算机的存储器是一维的，要把二维数组的元素存放到存储器中，pascal 是按行（第一个下标）的次序存放，即按 a [1，1] a [1，2] a [1，3] a [1，4] a [2，1] ...，a [5，4] 的次序存放于存储器中某一组连续的存储单元之内。

对于整个二维数组的元素引用时，大多采用二重循环来实现。如：给如上说明的二维数组 a 进行赋值:a[i,j]=i*j。

```
for i:=1 to 5 do
  for j:=1 to 4 do
      a[i,j]:=i*j;
```

对二维数组的输入与输出也同样可用二重循环来实现:

```
for i:=1 to 5 do
begin
  for j:=1 to 4 do
    read(a[i,j]);
  readln;
end;
for i:=1 to 5 do
  begin
    for j:=1 to 4 do
      write(a[i,j]:5);
    writeln;
  end;
```

3、多维数组的应用示例

例 1 设有一程序:

```
program ex5_3;
const n=3;
type matrix=array[1..n,1..n]of integer;
var a:matrix;
  i,j:1..n;
begin
  for i:=1 to n do
  begin
    for j:=1 to n do
      read(a[i,j]);
    readln;
  end;
for i:=1 to n do
```

```
begin
  for j:=1 to n do
    write(a[j,i]:5);
  writeln;
end;
end.
```

且运行程序时的输入为:

2□1□3←┘

3□3□1←┘

1□2□1←┘

则程序的输出应是:

2□3□1

1□3□2

3□1□1

例 2 输入 4 名学生数学、物理、英语、化学、pascal 五门课的考试成绩，求出每名学生的平均分，打印出表格。

分析：用二维数组 a 存放所给数据,第一下标表示学生的学号, 第二个下标表示该学生某科成绩，如 a[i,1]、a[i,2]、a[i,3]、a[i,4]、a[i,5]分别存放第 i 号学生数学、物理、英语、化学、pascal 五门课的考试成绩,由于要求每个学生的总分和平均分， 所以第二下标可多开两列，分别存放每个学生 5 门成绩和总分、平均分。

源程序如下：

```
program ex5_4;
var a:array[1..4,1..7]of real;
  i,j:integer;
begin
  fillchar(a,sizeof(a),0);
  {函数 fillchar 用以将 a 中所有元素置为 0}
  writeln('Enter 4 students score');
  for i:=1 to 4 do
 begin
    for j:=1 to 5 do {读入每个人 5 科成绩}
    begin
     read(a[i,j]);
     {读每科成绩时同时统计总分}
     a[i,6]:=a[i,6]+a[i,j];
    end;
    readln;
    a[i,7]:=a[i,6]/5;{求平均分}
 end;
 {输出成绩表}
 writeln( 'No. Mat. Phy. Eng. Che. Pas. Tot. Ave.');
 for i:=1 to 4 do
 begin
    write(i:2,' ');
    for j:=1 to 7 do
     write(a[i,j]:9:2);
    writeln;
 end;
end.
```

4、数组类型的应用

例 3 输入一串字符,字符个数不超过 100,且以"."结束。 判断它们是否构成回文。

分析：所谓回文指从左到右和从右到左读一串字符的值是一样的，如 12321,ABCBA,AA 等。先读入要判断的一串字符（放入数组 letter 中），并记住这串字符的长度，然后首尾字符比较，并不断向中间靠拢，就可以判断出是否为回文。

源程序如下：

```
program ex5_5;
var
    letter   :  array[1..100]of char;
    i,j      :  0..100;
    ch       :  char;
begin
   {读入一个字符串以'.'号结束}
  write('Input a string:');
  i:=0;read(ch);
  while ch<>'.' do
  begin
    i:=i+1;letter[i]:=ch;
    read(ch)
  end;
  {判断它是否是回文}
  j:=1;
  while (j<i)and(letter[j]=letter[i])do
  begin
    i:=i-1;j:=j+1;
  end;
  if j>=i  then writeln('Yes.')
  else writeln('No.');
end.
```

例 4　奇数阶魔阵

魔阵是用自然数 1，2，3...，n2 填 n 阶方阵的各个元素位置，使方阵的每行的元素之和、每列元素之和及主对角线元素之和均相等。奇数阶魔阵的一个算法是将自然数数列从方阵的中间一行最后一个位置排起，每次总是向右下角排（即 Aij 的下一个是 Ai+1,j+1）。但若遇以下四种情形，则应修正排数法。

（1） 列排完（即 j=n+1 时），则转排第一列;

（2） 行排完（即 i=n+1 时），则转排第一行;

（3） 对 An,n 的下一个总是 An,n-1;

（4） 若 Aij 已排进一个自然数，则排 Ai－1，j－2。

例如 3 阶方阵，则按上述算法可排成：

4 3 8
9 5 1
2 7 6

有了以上的算法，解题主要思路可用伪代码描述如下：

```
1  in div 2+1,yn /*排数的初始位置*/
2  a[i,j]1;
3  for k:=2 to nn do
4  计算下一个排数位置（i,j）;
5  if a[i,j]<>0 then
6  ii-1;
7  jj-2;
8  a[i,j]k;
9  endfor
```

对于计算下一个排数位置，按上述的四种情形进行，但我们应先处理第三处情况。算法描述如下：

```
1  if (i=n)and(j=n) then
2  jj-1; /*下一个位置为(n,n-1)*/;
3  else
4  ii mod n +1;
5  jj mod n +1;
6  endif;
```

源程序如下：

```
program ex5_7;
var
  a : array[1..99,1..99]of integer;
  i,j,k,n : integer;
begin
  fillchar(a,sizeof(a),0);
  write('n=');readln(n);
  i:=n div 2+1;j:=n;
  a[i,j]:=1;
  for k:=2 to n*n do
    begin
      if (i=n)and(j=n) then
        j:=j-1
      else
        begin
          i:=i mod n +1;
          j:=j mod n +1;
        end;
      if a[i,j]<>0 then
        begin
          i:=i-1;
          j:=j-2;
        end;
      a[i,j]:=k;
    end;
  for i:=1 to n do
    begin
      for j:=1 to n do
        write(a[i,j]:5);
      writeln;
    end;
end.
```

练习：

1、 输入N个同学的语、数、英三科成绩，计算他们的总分与平均分，并统计出每个同学的名次，最后以表格的形式输出。

2、 输出杨辉三角的前N行(N<10)。

```
    1
    1  1
    1  2  1
    1  3  3  1
    1  4  6  4  1
```

第七章 字符、字符串类型的使用

. 1 字符类型

字符类型为由一个字符组成的字符常量或字符变量。

字符常量定义：

```
const
  字符常量='字符'
```
字符变量定义：
```
Var
  字符变量：char;
```
字符类型是一个有序类型，字符的大小顺序按其ASCⅡ代码的大小而定。函数succ、pred、ord适用于字符类型。

例如：后继函数：succ('a')='b'
前继函数：pred（'B'）='A'
序号函数：ord（'A'）=65

例1 按字母表顺序和逆序每隔一个字母打印。即打印出：
```
a c e g I k m o q s u w y
z x r v t p n l j h f d b
```
程序如下：
```
program ex8_1;
  var letter：char;
  begin
    for letter：='a' to 'z' do
    if (ord(letter)-ord('a'))mod 2=0 then write(letter：3);
    writeln;
    for letter：='z' downto 'a' do
    if (ord(letter)-ord('z'))mod 2 =0 then write(letter：3);
  writeln;
end.
```
分析：程序中，我们利用了字符类型是顺序类型这一特性，直接将字符类型变量作为循环变量，使程序处理起来比较直观。

函数和过程名	功 能	说 明
copy(s,m,n)	取s中第m个字符开始的n个字符	若m大于s的长度，则返回空串；否则，若m+n大于s的长度，则截断
length(s)	求s的动态的长度	返回值为整数
pos(sub,s)	在s中找子串sub	返回值为sub在s中的位置，为byte型
insert(sour,s,m)	在s的第m个字符位置处插入子串sour	若返回串超过255，则截断
delete(s,m,n)	删除s中第m个字符开始的n个字符串	若m大于s的长度，则不删除；否则，若m+n大于s的长度，则删除到结尾
str(x[:w[:d]],s)	将整数或实数x转换成字符串s	w 和 d是整型表达式，意义同带字宽的write语句
val(s,x,code)	将字符串S 转换成整数或实数x	若S中有非法字符，则code存放非法字符在S中的下标；否则，code为零。code为整型
upcase(ch)	将字母ch转换成大写字母	若ch不为小写字母，则不转换

函数和过程名
功 能
说 明
copy(s,m,n)
取s中第m个字符开始的n个字符
若m大于s的长度，则返回空串；否则，若m+n大于s的长度，则截断
length(s)
求s的动态的长度
返回值为整数
pos(sub,s)

在 s 中找子串 sub
返回值为 sub 在 s 中的位置，为 byte 型
insert(sour,s,m)
在 s 的第 m 个字符位置处插入子串 sour
若返回串超过 255，则截断
delete(s,m,n)
删除 s 中第 m 个字符开始的 n 个字符串
若 m 大于 s 的长度，则不删除；否则，若 m+n 大于 s 的长度，则删除到结尾
str(x[:w[:d]],s)
将整数或实数 x 转换成字符串 s
w 和 d 是整型表达式，意义同带字宽的 write 语句
val(s,x,code)
将字符串 S 转换成整数或实数 x
若 S 中有非法字符，则 code 存放非法字符在 S 中的下标；否则，code 为零。code 为整型
upcase(ch)
将字母 ch 转换成大写字母
若 ch 不为小写字母，则不转换

例 4 ：校对输入日期(以标准英语日期,月/日/年)的正确性，若输入正确则以年．月．日的方式输出。

程序如下：

```
program ex8_4;
  const
    max:array[1..12] of byte
        =(31,29,31,30,31,30,31,31,30,31,30,31);
  var
    st:string;
    p,w,y,m,d:integer;
  procedure err;
    begin
      write('Input Error!');
      readln;
      halt;
    end;
  procedure init(var x:integer);
    begin
      p:=pos('/',st);
      if (p=0) or (p=1) or (p>3) then err;
      val(copy(st,1,p-1),x,w);
      if w<>0 then err;
      delete(st,1,p);
    end;
  begin
    write('The Date is :');
    readln(st);
    init(m);
    init(d);
    val(st,y,w);
    if not (length(st)<>4) or (w<>0) or (m>12) or (d>max[m]) then err;
    if (m=2) and (d=29)
      then if y mod 100=0
            then begin
                   if y mod 400<>0 then err;
                 end
```

```
      else if y mod 4<>0 then err;
    write('Date : ',y,'.',m,'.',d);
    readln;
  end.
```

分析：此题的题意很简单，但在程序处理时还需考虑以下几方面的问题。

1．判定输入的月和日应是 1 位或 2 位的数字，程序中用了一个过程 inst，利用串函数 pos，求得分隔符/所在的位置而判定输入的月和日是否为 1 位或 2 位，利用标准过程 val 判定输入的月和日是否为数字；

2．判定月和日是否规定的日期范围及输入的年是否正确；

3．若输入的月是 2 月份，则还需考虑闰年的情况。

例 5 ：对输入的一句子实现查找且置换的功能。

程序如下:

```
program ex8_5;
var
  s1,s,o:string;
  i:integer;
begin
  write('The text:');
  readln(s1);
  write('Find:');readln(s);
  write('Replace:');readln(o);
  i:=pos(s,s1);
  while i<>0 do begin
    delete(s1,i,length(s));
    insert(o,s1,i);
    i:=pos(s,s1);
  end;
  writeln(s1);
  readln;
end.
```

分析：程序中,输入要查找的字符串及要置换的字符串,充分用上了字符串处理的标准过程 delete、insert 及标准函数 pos。

. 2 字符串类型

字符串是由字符组成的有穷序列。

字符串类型定义：

```
type
<字符串类型标识符>=string[n];
```

```
var
  字符串变量：字符串类型标识符;
```

其中：n 是定义的字符串长度，必须是 0~255 之间的自然整数，第 0 号单元中存放串的实际长度，程序运行时由系统自动提供，第 1~n 号单元中存放串的字符。若将 string[n]写成 string，则默认 n 值为 255。

例如：

```
type
  man=string[8];
    line=string;
  var
    name: man;
    screenline: line;
```

另一种字符类型的定义方式为把类型说明的变量定义合并在一起。

例如：

```
var
    name: string[8];
    screenline: string;
```

Turbo Pascal 中，一个字符串中的字符可以通过其对应的下标灵活使用。

例如：var
I:integer;
name：string；
begin
readln（name）；
for i：=1 to ord（name[0]）do
writeln（name[i]）；
end.

语句 writeln（name[i]）输出 name 串中第 i 个字符。

例 2 求输入英文句子单词的平均长度.

程序如下：

```
program ex8_2;
  var
    ch:string;
    s,count,j:integer;
  begin
    write('The sentence is:');
    readln(ch);
    s:=0;
    count:=0;
    j:=0;
    repeat
      inc(j);
      if not (ch[j] in [':',',','.',';','''','!','?','.',' ']) then inc(s);
      if ch[j] in[' ','.','!','?'] then inc(count);
    until (j=ord（ch[0])) or (ch[j] in ['.','!','?']);
    if ch[j]<>'.' then writeln('It is not a sentence.')
    else writeln('Average length is ',s/count:10:4);
  end.
```

分析：程序中，变量 s 用于存句子中英文字母的总数，变量 count 用于存放句子中单词的个数，ch[j]表示 ch 串中的第 j 个位置上的字符,ord（ch[0]）为 ch 串的串长度。程序充分利用 Turbo Pascal 允许直接通过字符串下标得到串中的字符这一特点，使程序比较简捷。

. 3 字符串的操作

（一）字符串的运算和比较

由字符串的常量、变量和运算符组成的表达式称为字符串表达式。

字符串运算符包括：

1. +：连接运算符

例如：'Turbo '+'PASCAL'的结果是'Turbo PASCAL'。

若连接的结果字符串长度超过 255，则被截成 255 个字符。若连接后的字符串存放在定义的字符串变量中，当其长度超过定义的字符串长度时，超过部份字符串被截断。

例如：var
str1，str2，str3：string[8]；
begin
str1：='Turbo '；
str2：='PASCAL'；
str3：=str1+str2；
end.

则 str3 的值为：'Turbo PA'。

2. =、〈〉、〈、〈=、〉、〉=：关系运算符

两个字符串的比较规则为，从左到右按照 ASCⅡ码值逐个比较，遇到 ASCⅡ码不等时,规定 ASCⅡ码值大的字符所在的字符串为大。

例如：'AB'〈'AC' 结果为真；
'12'〈'2' 结果为真；

'PASCAL '='pascal' 结果为假；

例 3： 对给定的 10 个国家名，按其字母的顺序输出。

程序如下：

```
program ex8_3;
  var i,j,k:integer;
      t:string[20];
      cname:array[1..10] of string[20];
  begin
    for i:=1 to 10 do readln(cname[i]);
    for i:=1 to 9 do
      begin
        k:=i;
        for j:=i+1 to 10 do
          if cname[k]>cname[j] then k:=j;
        t:=cname[i];cname[i]:=cname[k];cname[k]:=t;
      end;
    for i:=1 to 10 do writeln(cname[i]);
  end.
```

分析：程序中，当执行到 if cname[k]>cname[j]时，自动将 cname[k]串与 cname[j]串中的每一个字符逐个比较，直至遇到不等而决定其大小。这种比较方式是计算机中字符串比较的一般方式。

. 4 字符串的函数和过程

Turbo Pascal 提供了八个标准函数和标准过程，见下表，利用这些标准函数与标准过程，一些涉及到字符串的问题可以灵活解决。

第八章 函数和过程

前面我们曾经学习了程序设计中的三种基本控制结构（顺序、分支、循环）。用它们可以组成任何程序。但在应用中，还经常用到子程序结构。

通常，在程序设计中，我们会发现一些程序段在程序的不同地方反复出现，此时可以将这些程序段作为相对独立的整体，用一个标识符给它起一个名字，凡是程序中出现该程序段的地方，只要简单地写上其标识符即可。这样的程序段，我们称之为子程序。

子程序的使用不仅缩短了程序，节省了内存空间及减少了程序的编译时间，而且有利于结构化程序设计。因为一个复杂的问题总可将其分解成若干个子问题来解决，如果子问题依然很复杂，还可以将它继续分解，直到每个子问题都是一个具有独立任务的模块。这样编制的程序结构清晰，逻辑关系明确，无论是编写、阅读、调试还是修改，都会带来极大的好处。

在一个程序中可以只有主程序而没有子程序(本章以前都是如此)，但不能没有主程序，也就是说不能单独执行子程序。pascal 中子程序有两种形式：函数和过程。

一、函数

在此之前，我们曾经介绍并使用了 pascal 提供的各种标准函数，如 ABS,SUCC 等等，这些函数为我们编写程序提供了很大的方便。但这些函数只是常用的基本函数，编程时经常需要自定义一些函数。

（一）函数的说明

在 pascal 中，函数也遵循先说明后使用的规则，在程序中，函数的说明放在调用该函数的程序（主程序或其它子程序）的说明部分。函数的结构主程序的结构很相似。

函数定义的一般格式:

function <函数名> (<形式参数表>):<类型>; {函数首部}

```
  <说明部分>
 begin                          函
       <语句>;                  数
        :                       体
       <语句>;
 end;
```

说明:

①函数由首部与函数体两部分组成。

②函数首部以关键字 function 开头。

③函数名是用户自定义的标识符。

④函数的类型也就是函数值的类型，所求得的函数值通过函数名传回调用它的程序。可见，函数的作用一般是为了求得一个值。

⑤形式参数简称形参，形参即函数的自变量。自变量的初值来源于函数调用。在函数中，形参一般格式如下：

变量名表 1：类型标识符 1；变量名表 2：类型标识符 2；...;变量名表 n：类型标识符 n

可见形参表相当于变量说明，对函数自变量进行说明，但应特别注意：此处只能使用类型标识符，而不能直接使用类型。

⑥当缺省形参表（当然要同时省去一对括号）时，称为无参函数。

⑦函数体与程序体基本相似，由说明部分和执行部分组成。

⑧函数体中的说明部分用来对本函数使用的标号、常量、类型、变量、子程序加以说明，这些量只在本函数内有效。

⑨函数体的执行部分由 begin 开头，end 结束，中间有若干用分号隔开的语句，只是 end 后应跟分号,不能像程序那样用句号".".。

⑩在函数体的执行部分，至少应该给函数名赋一次值，以使在函数执行结束后把函数值带回调用程序。

（二）函数的调用

我们可以在任何与函数值类型兼容的表达式中调用函数，或者说，函数调用只能出现在允许表达式出现的地方，或作为表达式的一个因子。

函数调用方式与标准函数的调用方式相同。

函数调用的一般格式：

＜函数名＞

或

＜函数名＞（实在参数表）

说明：

①实在参数简称实参。实参的个数必须与函数说明中形参的个数一致，实参的类型与形参的类型应当一一对应。

②调用函数时，一般的，实参必须有确定的值。

③函数调用的步骤为：计算实参的值，"赋给"对应的形参；

（三）函数的应用举例

例 1 求正整数 A 和 B 之间的完全数(A<B).

分析：所谓完全数是指它的小于该数本身的因子之和等于它本身，如 6＝1＋2＋3，6 即是一个完全数。因此我们可定义一个布尔型函数 perfect(x),若 x 是完全数,其值为 TURE,否则为 FALSE。整个程序算法如下：

```
1 for i:=A to B do
2 if perfect(i) then writeln(i);
```

源程序如下：

```
program ex7_1;
var
    i,a,b : integer;
function perfect(x:integer):boolean;
var
    k,sum : integer;
begin
  {累加 x 所有小于本身的因数}
  sum:=1;
  for k:=2 to x div 2 do
    if x mod k=0 then sum:=sum+k;
  {判断 x 是否是完全数}
  perfect:=x=sum; {将结果赋值给函数名}
end;{end of perfect}
```

```
begin{主程序开始}
  write('Input a,b:');
  repeat {输入 0<a<b}
    readln(a,b);
  until (a>0)and(b>0)and(a<b);
  writeln('List of all perfect numbers:');
  {从 a 到 b 逐个判断,是完全数则打印出来
  for i:=a to b do
                    函数的调用
    if perfect(i) then writeln(i);
end.
```

自定义函数只是主程序的说明部分，若主程序中没有调用函数，则系统不会执行函数子程序。当主程序调用一次函数时，则将实在参数的值传给函数的形式参数，控制转向函数子程序去执行，子程序执行完毕后自动返回调用处。

二、过程

在 pascal 中,自定义过程与自定义函数一样,都需要先定义后调用。函数一般用于求值，而过程一般实现某些操作。

（一）过程的说明

过程说明的一般格式为：

procedure <过程名> (<形式参数表>); {过程首部}

```
<说明部分>;
begin                          过
  <语句>;                      程
    :                          体
  <语句>;
end;
```

说明:

①过程首部以关键字 procedure 开头。

②过程名是用户自定义的标识符，只用来标识一个过程，不能代表任何数据，因此不能说明"过程的类型"。

③形参表缺省（当然要同时省去一对括号）时，称为无参过程。

④形参表的一般格式形式如下：

[var] 变量名表：类型；...；[var] 变量名表：类型。

其中带 var 的称为变量形参，不带 var 的称为值形参。在函数中，形参一般都是值形参，很少用变量形参（但可以使用）。例如，下列形参表中：

(x,y:real;n:integer;var w:real;var k:integer;b:real)

x、y、n、b 为值形参,而 w、k 为变量形参。

调用过程时，通过值形参给过程提供原始数据，通过变量形参将值带回调用程序。因此，可以说，值形参是过程的输入参数，变量形参是过程的输出参数。有关变参，这在后面内容具体叙述。

⑤过程体与程序、函数体类似。与函数体不同的是：函数体的执行部分至少有一个语句给函数名赋值，而过程体的执行部分不能给过程名赋值，因为过程名不能代表任何数据。

⑥过程体的说明部分可以定义只在本过程有效的标号、常量、类型、变量、子程序等。

（二）过程的调用

过程调用是通过一条独立的过程调用语句来实现的，它与函数调用完全不同。过程调用与调与标准过程（如 write,read 等）的方式相同。调用的一般格式为：

＜过程名＞

或

＜过程名＞（实在参数表）

说明: ①实参的个数、类型必须与形参一一对应。

②对应于值形参的实参可以是表达式，对应于变量形参的实参只能是变量。

③过程调用的步骤为：计算实参的值；将值或变量的"地址"传送给对应的形参；执行过程体；返回调用处。

过程与函数有下列主要区别：

①过程的首部与函数的首部不同；

②函数通常是为了求一个函数值，而过程可以得到若干个运算结果，也可用来完成一系列的数据处理，或用来完成与计算无关的各种操作；

③调用方式不同。函数的调用出现在表达式中，而过程调用是一个独立的语句。

（三）过程的应用举例

例 2 输出以下一个图形:

```
*
**
***
****
*****
******
```

分析：我们前面学习可用的二重循环打印出上图形, 现我们设置一个过程打印出 N 个连续的"*"号。

源程序如下：

```
program ex7_2;
  var i:integer;
  procedure draw_a_line(n:integer); {该过程打印出连续 n 个星号,并换行}
    var j:integer;
    begin
      for j:=1 to n do
        write('*');
      writeln;
    end;
begin
  for i:=1 to 6 do
    draw_a_line(i);{调用过程,第 I 行打印 i 个连续星号}
end.
```

三、过程、函数的数据传递

在程序调用子程序时，调用程序将数据传递给被调用的过程或函数，而当子程序运行结束后，结果又可以通过函数名、变参。当然也可以用全局变量等形式实现数据的传递。这一节我们，就来研究参数传递与局部变量、全局变量等问题。

（一）数值参数和变量参数

前面已经讲过，pascal 子程序中形式参数有数值形参（简称值参）和变量形参（变参）两种。事实上，还有函数形参和过程形参两种，只是应用并不太多，我们不作深入地研究。

1、值形参

值参的一般格式如§7.1.1 所示。应该强调的是：

①形参表中只能使用类型标识符，而不能使用类型。

②值形参和对应的实参必须一一对应，包括个数和类型。

③实参和值形参之间数据传递是单向的，只能由实参传送给形参，相当赋值运算。

④一个特殊情况是，当值形参是实型变量名时，对应的实参可以是整型表达式。

⑤值形参作为子程序的局部量，当控制返回程序后，值形参的存储单元释放。

2、变量形参

变量形参的一般格式如§7.2.1 所示，必须在形参前加关键字 var。

应该注意的是：

①与变量形参对应的实参只能是变量名，而不能是表达式。

②与变量形参对应的实参可以根据需要决定是否事先有值。

③变量形参与对应的实参的类型必须完全相同。

④对变量形参，运行时不另外开辟存储单元，而是与对应的实参使用相同的存储单元。也就是说，调用子程序时，是将实参的地址传送给对应的变量形参。

⑤当控制返回到调用程序后，变量形参的存储单元不释放，但变量形参本身无定义，即不得再使用。

⑥选用形式参时，到底是使用值形参还是变量形参，应慎重考虑。值形参需要另开辟存储空间，而变量形参会带来一些副作用。一般在函数中使用值形参，而在过程中才使用变量形参，但也有例外。

例 3 写出下列两个程序的运行结果。

```
program ex1;
  var a,b:integer;
  procedure swap(x,y:integer);
    var t:integer;
    begin
      t:=x;x:=y;y:=t;
    end;
  begin
    a:=1;b:=2;
    writeln(a:3,b:3);
    swap(a,b);
    writeln(a:3,b:3);
  end.
```

```
program ex2;
  var a,b:integer;
  procedure swap(Var x,y:integer) ;
      var t:integer;
       begin
         t:=x;x:=y;y:=t;
       end;
       begin
        a:=1;b:=2;
         writeln(a:3,b:3);
           swap(a,b);
           writeln(a:3,b:3);
        end.
```

分析：这两个程序唯一的区别是 ex1 中将 x,y 作为值形参，而 ex2 中将 x,y 作为变量形参，因此在 ex2 中对 x,y 的修改实际上是对调用该过程时与它们对应的变量 a,b 的修改，故最后，a,b 的值为 2，1。而 ex1 中调用 swap 过程时，只是将 a,b 的值传递给 x,y，之后在过程中的操作与 a,b 无关。

答：ex1 的运行结果为: ex2 的运行结果为:

```
 1 2                1 2
 1 2                2 1
```

（二）全程变量、局部变量及它们的作用域

在主程序的说明部分和子程序的说明部分均可以说明变量，但它们的作用范围是特定的。

1、局部量及其作用域

在介绍过程和函数的说明时，我们曾指出，凡是在子程序内部作用的变量，应该在本子程序内加以说明。这种在子程序内部说明的变量称为局部变量。形式参数也只是在该子程序中有效，因此也属于局部变量。

一个变量的作用域是指在程序中能对此变量进行存取的程序范围。因此，局部变量的作用域就是其所在的子程序。实际上，局部变量只是当其所在的子程序被调用时才具有确定的存储单元，当控制从子程序返回到调用程序后，局部变量的存储单元就被释放，从而变得无定义。

事实上，在子程序内定义的标号、符号常量、类型、子程序也与局部变量具有相同的作用域。

2、全程量及其作用域

全程量是指在主程序的说明部分中说明的量。全程量的作用域分两种情况：

①当全程量和局部量不同名时，其作用域是整个程序范围（自定义起直到主程序结束）。

②当全程量和局部量同名时，全程量的作用域不包含局部量的作用域。

例 4 写出下列程序的运行结果:

```
program ex7_4;
  var x,y:integer;
  procedure a;
    var x:integer;
    begin
      x:=2;
      writeln('#',x,'#');
      writeln('#',y,'#');
    end;{of a}
  begin{main program}
    x:=1;y:=2;
    writeln('*',x,'*', y);
```

```
    a;
    writeln('***',x,'***',y);
  end.
```

分析：程序中 x,y 是全局变量,但在过程 a 中也有变量 x，故全程变量 x 的作用域为除过程 a 外的任何地方。而 y 的作用域包含了子程序 a，即整个程序。

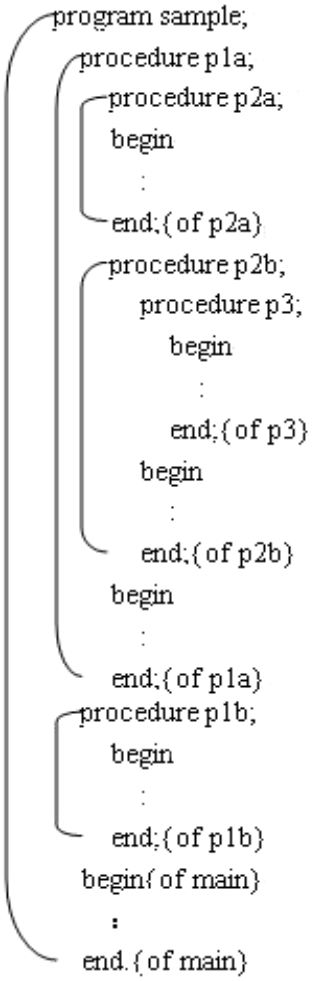
答：运行结果如下:

```
*1*2
#2#
#2#
***1***2
```

评注：变量作用域内对变量的操作都是对同一存储单元中的量进行的。

四、过程和函数的嵌套

Pascal 语言中，使用过程和函数，能使程序设计简短，便于阅读，节省存贮单元和编译时间。程序往往设计成分层结构，由一个主程序和若干个过程及函数组成。在过程或函数中，还可以说明另一些过程或函数，即过程或函数可以分层嵌套。在同一层中亦可说明几个并列的过程或函数。例如：



上例过程的分层嵌套关系如下：0 层主程序 sample 内并列两个 1 层过程 P1a 和 P1b。过程 P1a 又嵌套两个 2 层过程 p2a 和 p2b，2 层的第二过程 p2b 又嵌套过程 p3，p3 就是第 3 层。其中 p1b,p2a 和 p3 不再嵌套别的过程，称为基本过程。这种分层结构的程序设计，特别要注意局部变量的使用范围和过程调用的要求。

在主程序 sample 中定义的变量,可以在所有的过程中使用,主程序可调用 p1a 和 p1b 两个过程。过程 p1a 中定义的变量,只能在 p2a,p2b 和 p3 中使用。它能调用 p2a,p2b 两个过程,而不能调用 p3 和 p1b。 在过程 p1b 中定义的变量,只能在 p1b 中使用,它只能调用过程 p1a。过程 p2a 不能调用任何过程。过程 p2b 可以调用并列过程 p2a 和 p3，而过程 p3 可以调用 p2a 过程。

过程调用是有条件的，过程定义在先，调用在后。同一层过程，后说明的过程可以调用先说明的过程。如果要调用在它后面定义的过程（或函数），可使用<向前引用>FORWARD 这个扩充标识符。要注意的是<向前引用>过程(或函数)首部中形式参数表写一次即可, 不必重复。如：

```
procedure extend(var a,b:integer);
forward;
```

表示过程 extend<向前引用>。因此,过程 extend 的说明部分只须如下书写:

```
procedure extend;
<说明部分>
begin
:
end;
```

五、子程序（模块化）结构的程序设计

例 5 对 6 到 60 的偶数验证哥德巴赫猜想：不小于 6 的偶数可分解成两个素数之和。

分析：用布尔型函数 prime(x)判断 x 是否是素数,若是, 函数值为真，否则，函数值为假。算法如下所示。

```
1.  t:=6
2.  while t≤60 do
3.  t11;
4.  repeat
5.  t11+2; /* 找下一个素数 a */
6.  until prime(t1)and prime(t-t1); /*直到 a,b 都是素数*/
7.  writeln(i,'=',t1,'+',t-t1);
8.  tt+2;
9.  endwhile
```

源程序如下:

```
program ex9_7;
  var t,t1:integer;
  function prime(x:integer):boolean;
    var i:integer;
    begin
      if x=1
        then prime:=false
        else if x=2
                then prime:=true
                else begin
                       prime:=true;
                       i:=2;
                       while (i<=round(sqrt(x)))and(x mod i<>0) do
                         i:=i+1;
                         if i<=round(sqrt(x)) then prime:=false;
                     end;
    end;{of prime}
    begin
      t:=6;
      while t<=60 do
        begin
          t1:=1;
          repeat
            t1:=t1+2;
          until prime(t1) and prime(t-t1);
          writeln(t,'=',t1,'+',t-t1);
          t:=t+2;
        end;
```

```
  end.
```

例 6 编写一个给一个分数约分的程序。

源程序如下：

```
program ex7_6;      ┌———变量参数
   var a,b:integer; ↓
    procedure common(var x,y:integer);
      var i,j,k:integer;
      begin
        {求 x,y 的最大公约数}
        i:=x;j:=y;
        repeat
          k:=i mod j;
          i:=j;j:=k;
        until k=0;
          {对 x,y 进行约分}
            x:=x div i;y:=y div i;
        end;
      begin
          write('Input a,b=');readln(a,b);
          common(a,b);
          writeln(a,b:5);
        end.
```

如输入:

```
Input a,b=12 8
```

则输出:

```
3 2
```

练习

1. 输入 5 个正整数求它们的最大公约数。（提示：可用一个数组将 5 个数存放起来，然后求第一个数和第二个数的公约数，再求第三个数与前两个数公约数的公约数，这样求得前三个整数最大公约数.....如此类推可求出 5 个整数的最大公约数）

2. 给一维数组输入任意 6 个整数,假设为:

   7 4 8 9 1 5

   请建立一个具有以下内容的方阵：

   ```
   7 4 8 9 1 5
   4 8 9 1 5 7
   8 9 1 5 7 4
   9 1 5 7 4 8
   1 5 7 4 8 9
   5 7 4 8 9 1
   ```

   (请用子程序编写)。

3. 求两个正整数的最小公倍数。

4. 输入一个任意位的正整数,将其反向输出。

5. 有五位同学,其各科成绩如下:

学号	数学	语文	英语	总分	名次
1	108	97	90		
2	98	88	100		
3	100	43	89		
4	84	63	50		
5	97	87	100		

（1）编写一个过程 enter,输入每个学生成绩并计算各人的总分。

（2）编写过程 minci,用以排出每个人的名次。

（3）按学号顺序输出。

第九章 集合类型和记录类型

一、集合类型

集合是由具有某些共同特征的元素构成的一个整体。在 pascal 中,一个集合是由具有同一有序类型的一组数据元素所组成，这一有序类型称为该集合的基类型。

（一）集合类型的定义和变量的说明

集合类型的一般形式为：

```
type 集合类型名＝set of <基类型>;
```

说明:

①基类型可以是任意顺序类型, 而不能是实型或其它构造类型。同时，基类型的数据的序号不得超过 255。例如下列说明是合法的:

```
type letters=set of 'A'..'Z';
     numbers=set of 0..9;
     s1=set of char;
     ss=(sun,mon,tue,wed,thu,fri,sat);
     s2=set of ss;
```

②与其它自定义类型一样, 可以将类型说明与变量说明合并在一起.如:

```
type numbers=set of 0..9;
var s:numbers;
```

与 var s:set of 0..9;等价。

（二）集合的值

集合的值是用"["和"]"括起来，中间为用逗号隔开的若干个集合的元素。如：

[] 空集

[1,2,3]

['a','e','i','o','u']

都是集合。

说明：

①集合的值放在一对方括号中，各元素之间用逗号隔开。

②在集合中可以没有任何元素，这样的集合称为空集。

③在集合中，如果元素的值是连续的，则可用子界型的表示方法表示。例如：

[1,2,3,4,5,7,8,9,10,15]

可以表示成：

[1..5,7..10,15]

④集合的值与方括号内元素出现的次序无关。例如，[1,5,8] 和 [5,1,8] 的值相等。

⑤在集合中同一元素的重复出现对集合的值没有影响。例如，[1,8,5,1,8] 与 [1,5,8] 的值相等。

⑥每个元素可用基类型所允许的表达式来表示。如 [1,1+2,4]、[ch]、[succ(ch)]。

（三）集合的运算

１、赋值运算

只能通过赋值语句给集合变量赋值，不能通过读语句赋值，也不能通过 write(或 writeln)语句直接输出集合变量的值。

２、集合的并、交、差运算

可以对集合进行并、交、差三种运算，每种运算都只能有一个运算符、两个运算对象，所得结果仍为集合。三种运算符分别用"＋"、"＊"、"－"表示。注意它们与算术运算的区别。

３、集合的关系运算

集合可以进行相等或不相等、包含或被包含的关系运算，还能测试一个元素是否在集合中。所用的运算符分别是：＝、＜＞、＞＝、＜＝、ｉｎ

它们都是二目运算，且前 4 个运算符的运算对象都是相容的集合类型，最后一个运算符的右边为集合，左边为与集合基类型相同的表达式。

例 4 设有如下说明：

```
type weekday=(sun,mon,tue,wed,thu,fri,sat);
     week=set of weekday;
     subnum=set of 1..50;
```

写出下列表达式的值:

⑴ [sun,sat] + [sun,tue,fri]
⑵ [sun,fri] * [mon,tue]
⑶ [wun,sat] * [sun..sat]
⑷ [sun] - [mon,tue]
⑸ [mon] - [mon,tue]
⑹ [sun..sat] - [mon,sun,sat]
⑺ [1,2,3,5] = [1,5,3,2]
⑻ [1,2,3,4] <> [1..4]
⑼ [1,2,3,5] >= [1..3]
⑽ [1..5] <= [1..4]
⑾ [1,2,3] <= [1..3]
⑿ 2 in [1..10]
答: 表达式的值分别是:
⑴ [sun,sat,tue,fri]
⑵ [ ]
⑶ [sun,sat]
⑷ [ ]
⑸ [ ]
⑹ [tue..fri]
⑺ TRUE
⑻ FALSE
⑼ TRUE
⑽ FALSE
⑾ TRUE
⑿ TRUE
例 5 输入一系列字符，对其中的数字字符、字母字符和其它字符分别计数。输入'？'后结束。
源程序如下：

```
program ex10_2;
var id,il,io:integer;
    ch:char;
    letter:set of char;
    digit:set of '0'..'9';
begin
    letter=['a'..'z','A'..'Z'];
    digit:=['0'..'9'];
    id:=0;il:=0;io:=0;
    repeat
     read(ch);
     if ch in letter
     then il:=il+1
     else if ch in digit
        then id:=id+1
        else io:=io+1;
  until ch='?';
 writeln('letter:',il,'digit:',id,'Other:',io);
end.
```

二、记录类型
在程序中对于组织和处理大批量的数据来说，数组是一种十分方便而又灵活的工具，但是数组在使用中有一个基本限制，这就是：一个数组中的所有元素都必须具有相同的类型。但在实际问题中可能会遇到另一类数据，它是由性质各不相同的成份组成的，即它的各个成份可能具有不同的类型。例如，有关一个学生的数据包含下列项目：
学号　　字符串类型
姓名　　字符串类型

年龄　　整型
性别　　字符型
成绩　　实型数组

Pascal 给我们提供了一种叫做记录的结构类型。在一个记录中，可以包含不同类型的并且互相相关的一些数据。

（一）记录类型的定义

在 pascal 中，记录由一组称为"域"的分量组成，每个域可以具有不同的类型。

记录类型定义的一般形式：

```
type 记录类型名＝record
 <域名 1>:<类型 1>;
 <域名 2>:<类型 2>;
  : :
  : :
  <域名 n>:<类型 n>;
end;
```

说明：

①域名也称域变量标识符， 应符合标识符的语法规则。在同一个记录中类型中，各个域不能取相同的名，但在不同的记录类型中，两个类型中的域名要以相同。

②记录类型的定义和记录变量可以合并为一个定义，如：

```
type date=record
        year:1900..1999;
        month:1..12;
        day:1..31
      end;
var x:date;
```

可以合并成:

```
var x: record
        year:1900..1999;
        month:1..12;
        day:1..31
      end;
```

③对记录的操作,除了可以进行整体赋值，只能对记录的分量——域变量进行。

④域变量的表示方法如下：

记录变量名.域名

如前面定义的记录 X,其 3 个分量分别为:x.year ， x.month ， x.day。

⑤域变量的使用和一般的变量一样, 即域变量是属于什么数据类型，便可以进行那种数据类型所允许的操作。

（二）记录的嵌套

当一个记录类型的某一个域类型也是记录类型的时候，我们说发生了记录的嵌套，看下面的例子：

例 6 某人事登记表可用一个记录表示, 其中各项数据具有不同的类型，分别命名一个标识符。而其中的"出生年月日"又包括三项数据，还可以用一个嵌套在内层的记录表示。

具体定义如下:

```
type  sexs=(male,female);
      date=record
        year:1900..1999;
        month:1..12;
        day:1..31;
      end;
personal=record
        name:string[15];
        sex:sexs;
        birthdate:date;
```

```
        home:string[40];
      end;
```

例 7 设计一个函数比较两个 dates 日期类型记录变量的迟早。

设函数名、形参及函数类型定义为：

```
AearlyB(A,B:dates):boolean;
```

函数的形参为两个 dates 类型的值参数。当函数值为 true 时表示日期 A 早于日期 B,否则日期 A 迟于日期 B 或等于日期 B。显然不能对 A 、B 两个记录变量直接进行比较，而要依具体的意义逐域处理。

源程序如下：

```
program ex6_7;
type dates=record
       year:1900.1999;
       month:1..12;
       day:1..31
     end;
var x,y:dates;
function AearlyB(A,B:dates):boolean;
var earln:boolean;
begin
  early:=false;
  if (A.year<B.year) then early:=true;
  if (A.year=B.year)and(A.month<B.month)
  then early:=true;
  if (A.year=B.year)and(A.month=B.month)and(A.day<B.day)
  then early:=true;
  AearlyB:=early;
  end;{of AearlyB}
begin
  write('Input DATE X(mm-dd-yy):')readln(X.month,X.day,X.year);
  write('Input DATE Y(mm-dd-yy):')readln(Y.month,Y.day,Y.year);
    if AearlyB(X,Y) then writeln(Date X early!') else writeln('Date X not early!');
end.
```

（三）开域语句

在程序中对记录进行处理时，经常要引用同一记录中不同的域，每次都按 6.4.1 节所述的格式引用，非常乏味。为此 Pascal 提供了一个 with 语句，可以提供引用域的简单形式。

开域语句一般形式:

```
with <记录变量名表> do
<语句>
```

功能: 在 do 后的语句中使用 with 后的记录的域时, 只要直接写出域名即可，即可以省略图 10.2.2 中的记录变量名和"."。

说明: ①一般在 with 后只使用一个记录变量名。如：

```
write('Input year:');
readln(x.year);
write('Input month:');
readln(x.month);
write('Input day:');
readln(x.day);
```

可以改写成:

```
with x do
  begin
    write('Input year:');readln(year);
    write('Input month:');readln(month);
```

```
    write('Input day:');readln(day);
  end;
```
②设 x,y 是相同类型的记录变量，下列语句是非法的：
```
    with x,y do...;
```
③with 后接若干个记录名时,应是嵌套的关系。如有记录说明：
```
    var x:record
        i:integer;
        y:record
            j:0..5;
            k:real;
          end;
        m:real
      end;
```
可以使用:
```
with x do
begin
    read(i);
    with y do
        read(j,k);
    readln(m);
end;
```
或简写为:
```
with x,y do
readln(i,j,k,m);
```
例 8 读入 10 个日期，再对每个日期输出第二天的日期。输入日期的格式是月、日、年，如 9□30□1993，输出的格式为 10/1/1993。

分析: 可用一个记录变量 today 表示日期。 知道一个日期后要更新为第二天的日期，应判断输入的日期是否为当月的最后一天，或当年的最后一天。

源程序如下:
```
program ex6_8;
  type date=record
            month:1..12;
            day:1..31;
            year:1900..1999;
          end;
  var today:array[1..10]of date;
        i:integer;
        maxdays:28..31;
  begin
    for i:=1 to 10 do {输入 10 个日期}
      with today[i] do
        readln(month,day,year);
    for i:=1 to 10 do
      with today[i] do{求第 i 个日期中月份最后一天 maxdays}
        begin
          case month of
            1,3,5,7,8,10,12:maxdays:=31;
            4,6,9,11 :maxdays:=30;
            2          :if(year mod 400=0) or( year mod 4=0)
                          and(year mod 100<>0)
                        then maxdays:=29
                        else maxdays:=28;
          end;
```

```
          if day=maxdays
            then begin
                day:=1;
                if month=12
                  then begin
                            month:=1;year:=year+1;
                       end
                    else month:=month+1;
                end
             else day:=day+1;
          writeln(month,'/',day,'/',year);
      end;
end.
```

三、应用实例

例 9 编制用筛法求 1-n(n≤200)以内素数的程序。

分析：由希腊著名数学家埃拉托色尼提出的所谓"筛法"，步骤如下：

①将所有候选数放入筛中；

②找筛中最小数（必为素数）next，放入集合 primes 中；

③将 next 的所有倍数从筛中筛去；

④重复②～④直到筛空。

编程时，用集合变量 sieve 表示筛子，用集合 primes 存放所有素数。

源程序如下：

```
program ex10_3;
const n=200;
var sieve,primes:set of 2..n;
    next,j:integer;
begin
    sieve:=[2..n];{将所有候选数放入筛中}
    primes:=[];{素数集合置空}
    next:=2;
    repeat
      {找筛 sieve 中最小一个数}
      while not(next in sieve) and(next<=n)do
        next:=succ(next);
      primes:=primes+[next];{将最小数放入素数集合中}
      {将这个素数的倍数从筛中删去}
    j:=next;
    while j<=n do
      begin
        sieve:=sieve-[j];
        j:=j+next;
      end
    until sieve=[];
    j:=0;
    for next:=2 to n do{打印出所有素数}
      if next in primes then
      begin
        write(next:5);
        j:=j+1;
        if j mod 10=0 then writeln;
      end;
    writeln;
end.
```

练习

1．一家水果店出售四种水果，每公斤的价格是：苹果 1.50 元，桔子 1.80 元，香蕉 2.0，菠萝 1.60 元。编一个程序，使售货员只要从键盘输入货物的代码及重量，计算机便能显示货物的名称、单价、重量及总价。

2．输入一个英语句子，以句号．为结束标志， 统计句子中元音字母出现的次数，把句子所有辅音字母组成一个集合，并把这些辅音字母打印出来。

3．编程序建立某班 25 人的数学课程成绩表，要求用数组类型和记录类型，其成绩表格式如下：

姓名	性别	平时成绩	期中考试	期终考试	总评成绩
张良	男	90	85	92	?
王心	男	70	82	71	?
……					
李英	女	82	84	75	?

其中总评成绩=平时成绩×20％＋期中考试×30％＋期终考试×％50。

4． 输入五个学生的出生日期(月/日/年)和当天的日期,然后用计算机计算出每个人到当天为止的年龄各是多少?（如某人 1975 年 10 月 1 日出生,今天是 94 年 12 月 1 日,则他的年龄应为 19 岁,而另一人的出生日期为 76 年 12 月 30 日,则他的年龄为 17 岁。）

第十章 指针

11.1 指针的动态变量

1.定义指针类型

在 Turbo Pascal 中，指针变量中存放的某个存储单元的地址，即指针变量指向某个存储单元。一个指针变量仅能指向某一种类型的存储单元，这种数据类型是在指针类型的定义中确定的，称为指针类型的基类型。指针类型定义如下：

类型名＝^基类型名;

例如：type q=^integer;

var a,b,c:q;

说明 q 是一指向整型存储单元的指针类型，其中"^"为指针符。a,b,c 均定义为指针变量，分别可以指向一个整型存储单元。

上例也可定义为：

var a,b,c:^integer;

指针也可以指向有结构的存储单元。

例如：
```
type person=record
            name:string[10];
            sex:(male,female);
            age:20..70
        end;
var pt:^person;
```
pt 为指向记录类型 person 的指针变量。

2.动态变量

应用一个指针指向的动态存储单元即动态变量的形式如下：

指针变量名^

例如：p^、q^、r^

指针变量 p 和它所指向的动态变量^p 之间有如下关系：

P—>P'

以下语句把整数 5 存放到 p 所指向的动态变量 p^ 中去：

p^:=5;

以下语句把 p 所指向的 p^中的值赋给整型变量 i：

i:=p^;

如果指针变量 p 并未指向任何存储单元，则可用下列赋值语句：

p:=nil;

其中 nil 是 Turbo Pascal 保留字，表示"空"，相当于 C 里面的 null

11.2 对动态变量的操作

在 Turob Pascal 程序中，动态变量不能由 var 直接定义而是通过调用标准过程 new 建立的。过程形式为：

new(指针变量名);

如果有下列变量定义语句：

```
var p:^integer;
```

仅仅说明了 p 是一个指向整型变量单元的指针变量，但这个整型单元并不存在，在指针变量 p 中还没有具体的地址值。在程序中必须通过过程调用语句：new(p);才在内存中分配了一个整型变量单元，并把这个单元的地址放在变量 p 中，一个指针变量只能存放一个地址。在同一时间内一个指针只能指向一个变量单元。当程序再次执行 new(p)时，又在内存中新建立了一个整型变量单元，并把新单元的地址存放在 p 中，从而丢失了旧的变量单元的地址。

为了节省内存空间，对于一些已经不使用的现有动态变量，应该使用标准过程 dispose 予以释放。过程形式为：dispose(指针变量名)；为 new(指针变量名)的逆过程，其作用是释放由指针变量所指向的动态变量的存储单元。例如在用了 new(p)后在调用 dispose(p)，则指针 p 所指向的动态变量被撤销，内存空间还给系统，这时 p 的值为 nil。

例：输入两个数，要求先打印大数后打印小数的方式输出，用动态变量做。

```
program dongtai;
  type intepter=^integer;
  var p1,p2:intepter;
  procedure swap(var,q1,q2:intepter);
  var p:integer;
begin
  p:=q1;q1:=q2;q2:=p;
end;
begin
  new(p1);new(p2);
  writeln('input 2 data: ');readln(p1^,p2^);
  if p1^    writeln('output 2 data: ',p1^:4,p2^:$);
end.
```

前面介绍的各种简单类型的数据和构造类型的数据属于静态数据。在程序中，这些类型的变量一经说明，就在内存中占有固定的存储单元，直到该程序结束。

程序设计中，使用静态数据结构可以解决不少实际问题，但也有不便之处。如建立一个大小未定的姓名表，随时要在姓名表中插入或删除一个或几个数据。而用新的数据类型——指针类型。通过指针变量，可以在程序的执行过程中动态地建立变量，它的个数不再受限制，可方便地高效地增加或删除若干数据。

一、指针的定义及操作

（一）指针类型和指针变量

在 pascal 中,指针变量(也称动态变量)存放某个存储单元的地址；也就是说， 指针变量指示某个存储单元。

指针类型的格式为:type 类型名＝^基类型名

说明:

①一个指针只能指示某一种类型数据的存储单元，这种数据类型就是指针的基类型。基类型可以是除指针、文件外的所有类型。例如，下列说明：

```
type pointer=^Integer;
var p1,p2:pointer;
```

定义了两个指针变量 p1 和 p2，这两个指针可以指示一个整型存储单元(即 p1、p2 中存放的是某存储单元的地址，而该存储单元恰好能存放一个整型数据)。

②和其它类型变量一样，也可以在 var 区直接定义指针型变量。

例如：var a:^real; b:^boolean;

又如：

```
type person=record
                  name:string[20];
                  sex:(male,female);
                  age:1..100
                end;
```

```
var pts:^person;
```

③pascal 规定所有类型都必须先定义后使用，但只有在定义指针类型时可以例外，如下列定义是合法的：

```
type pointer=^rec;
       rec=record
           a:integer;
           b:char
         end;
```

（二）开辟和释放动态存储单元

1、开辟动态存储单元

在 pascal 中，指针变量的值一般是通过系统分配的，开辟一个动态存储单元必须调用标准过程 new。

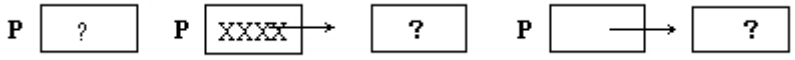
new 过程的调用的一般格式:

New(指针变量)

功能：开辟一个存储单元，此单元能存放的数据的类型正好是指针的基类型，并把此存储单元的地址赋给指针变量。

说明：

①这实际上是给指针变量赋初值的基本方法。例如，设有说明：var p:^Integer;
这只定义了 P 是一个指示整型存储单元的指针变量，但这个单元尚未开辟，或者说 P 中尚未有值（某存储单元的首地址）。当程序中执行了语句 new(p)才给 p 赋值，即在内存中开辟（分配）一个整型变量存储单元，并把此单元的地址放在变量 p 中。示意如下图：



(a)编译时给 p 分配空间，?表示值不定　(b)执行 New(p)后生成新单元，新单元的地址为 XXXX　(c)(b)的简略表示

内存单元示意图

②一个指针变量只能存放一个地址。如再一次执行 New(p)语句，将在内存中开辟另外一个新的整型变量存储单元，并把此新单元的地址放在 p 中，从而丢失了原存储单元的地址。

③当不再使用 p 当前所指的存储单元时，可以通过标准过程 Dispose 释放该存储单元。

2.释放动态存储单元

dispose 语句的一般格式:dispose(指针变量)

功能：释放指针所指向的存储单元，使指针变量的值无定义。

（三）动态存储单元的引用

在给一个指针变量赋以某存储单元的地址后，就可以使用这个存储单元。

引用动态存储单元一般格式：＜指针变量＞^

说明：

①在用 New 过程给指针变量开辟了一个它所指向的存储单元后，要使用此存储单元的唯一方法是利用该指针。

②对动态存储单元所能进行的操作是该类型（指针的基类型）所允许的全部操作。

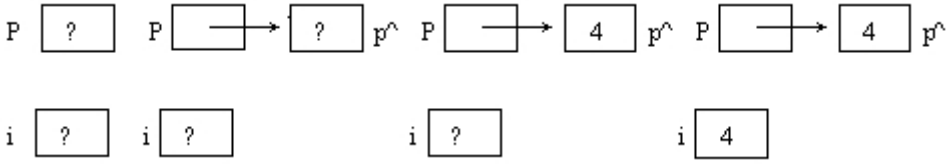
例 1 设有下列说明:

```
var p:^integer; i:integer;
```

画出执行下列操作后的内存示意图：

```
New(p); P^:=4;i:=p^;
```

解: 如下图所示。



(a)编译时分配存储单元　(b)执行 New 语句　(c)执行 P^:=4　(d)执行 i:=P^

内存单元示意图

（四）对指针变量的操作

前已述及，对指针所指向的变量（如Ｐ＾）可以进行指针的基类型所允许的　全部操作。对指针变量本身，除可用 New、Dispose 过程外，尚允许下列操作：

１、具有同一基类型的指针变量之间相互赋值

例 2 设有下列说明与程序段:

```
var p1,p2,p3:^integer;
begin
 New(P1) ; New(P2); New(P3);
  P1:=P2; P2:=P3;
end;
```

2、可以给指针变量赋 nil 值

nil 是 PASCAL 的关键字，它表示指针的值为"空"。例如，执行：

p1:=nil 后，p1 的值是有定义的，但 p1 不指向任何存储单元。

3、可以对指针变量进行相等或不相等的比较运算

在实际应用中，通常可以在指针变量之间，或指针变量与 nil 之间进行相等(＝)或不相等(＜＞＝的比较，比较的结果为布尔量。

例 3 输入两个整数，按从小到大打印出来。

分析：不用指针类型可以很方便地编程，但为了示例指针的用法，我们利用指针类型。定义一个过程 swap 用以交换两个指针的值。

源程序如下：

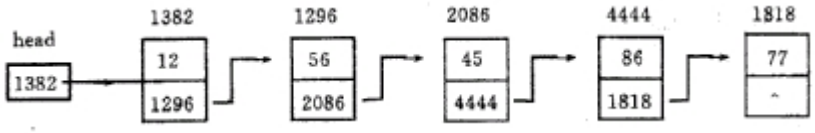
```
Type pointer=^integer;
var p1,p2:pointer;
procedure swap(var q1,q2:pointer);
  var q:pointer;
  begin
    q:=q1;
    q1:=q2;
    q2:=q;
  end;
begin
  new(p1);new(p2);
  write('Input 2 data:');readln(pq^,p2^);
  if p1^>p2^ then swap(p1,p2);
  writeln('Output 2 data:',p1^:4,p2^:4);
end.
```

二、链表结构

设有一批整数(12，56，45，86，77，.....，)，如何存放呢? 当然我们可以选择以前学过的数组类型。但是，在使用数组前必须确定数组元素的个数。如果把数组定义得大了，就会有大量空闲存储单元，定义得小了，又会在运行中发生下标越界的错误，这是静态存储分配的局限性。

利用本章介绍的指针类型可以构造一个简单而实用的动态存储分配结构－－链表结构。

下图是一个简单链表结构示意图：



其中：①每个框表示链表的一个元素，称为结点。

②框的顶部表示了该存储单元的地址(当然，这里的地址是假想的)。

③每个结点包含两个域：一个域存放整数，称为数据域，另一个域存放下一个结点(称为该结点的后继结点，相应地，该结点为后继结点的前趋结点)的地址。

④链表的第一个结点称为表头，最后一个结点表尾，称为指针域；

⑤指向表头的指针 head 称为头指针(当 head 为 nil 时，称为空链表)，在这个指针变量中 存放了表头的地址。

⑥在表尾结点中，由指针域不指向任何结点，一般放入 nil。

（一）链表的基本结构

由上图可以看出：

①链表中的每个结点至少应该包含两个域；一是数据域，一是指针域。因此，每个结点都是一个记录类型，指针的基类型也正是这个记录类型。因此，head 可以这样定义：

```
type pointer=^ rec;
  rec=record
        data: integer;
        next:pointer;
      end;
  var head:pointer;
```

②相邻结点的地址不一定是连续的。整个链表是通过指针来顺序访问的，一旦失去了一个指针值，后面的元素将全部丢失。

③与数组结构相比，使用链表结构时；可根据需要采用适当的操作步骤使链表加长或缩 短，而使存储分配具有一定的灵活性。这是链表结构的优点。

④与数组结构相比，数组元素的引用比较简单，直接用"数组名[下标]"即可，因为数组元素占用连续的存储单元，而引用链表元素的操作却比较复杂。

（二）单向链表的基本操作

上图所示的链表称为单向链表。下面我们通过一些例题来说明对单向链表的基本操作，并假设类型说明如前所述。

例 6 编写一个过程，将读入的一串整数存入链表， 并统计整数的个数。

分析：过程的输入为一串整数，这在执行部分用读语句完成。过程的输出有两个：一是链表的头指针，一是整数的个数，这两个输出可以用变量形参来实现。

由于不知道整数的个数，我们用一个特殊的 9999 作为结束标记。

过程如下：

```
procedure creat(var h:pointer;var n:integer);
var p,q:pointer;x:integer;
begin
    n:=0;h:=nil; read(x);
    while x<>9999 do
    begin
      New(p);
      n:=n+1;p^.data:=x;
      if n=1 then h:=p
      else q^.next:=p;
      q:=p;read(x)
    end;
    if h<>nil then q^.next:=nil;
    Dispose(p);
end;
```

例 7 编一过程打印链表 head 中的所有整数,5 个一行。

分析：设置一个工作指针 P,从头结点顺次移到尾结点，每移一次打印一个数据。

过程如下：

```
procedure print(head:pointer);
var p:pointer; n:integer;
begin
  n:=0;p:=head;
  while p<>nil do
    begin
      write(p^.data:8);n:=n+1;
      if n mod 5=0 then writeln;
```

```
      p:=p^.next;
    end;
  writeln;
end;
```

（三）链表结点的插入与删除

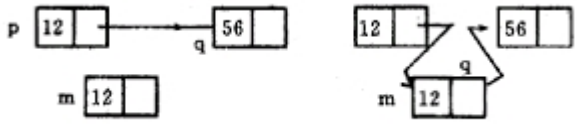
链表由于使用指针来连接，因而提供了更多了灵活性，可以插入删除任何一个成分。

设有如下定义：

```
type pointer=^rec;
rec=record
    data:integer;
    next:pointer
  end;
var head:pointer;
```

1、结点的插入

如下图所示,要在 P 结点和 Q 结点之间插入一个结点 m,其操作如下:



只要作如下操作即可:

```
  New(m);
  read(m^.data);
  m^.next:=q;
  p^.next:=m;
```

例 8 设链表 head 中的数据是按从小到大顺序存放的，在链表中插入一个数，使链表仍有序。

分析：显然，应分两步：查找、插入。设 p o 指向要插入的结点，若仅知道 p o 应插在 p 之前（作为 p 的前趋结点）是无法插入的，应同时知道 p 的前趋结点地址 q 。

当然，如果插在链表原头结点这前或原链表为空表或插在原尾结点之后，则插入时又必须作特殊处理。

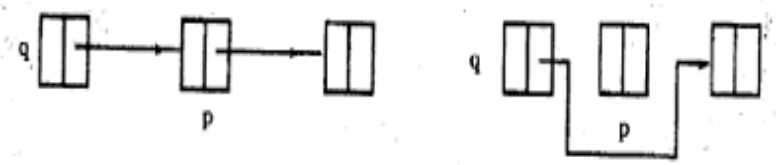
过程如下：

```
procedure inserting(var head:pointer;x:integer);
var po,p,q:pointer;
begin
  new(po);po^.data:=x;
  p:=head;
  if head=nil{原表为空表}
    then begin
      head:=po;po^.next:=nil;
    end
  else begin
    while (p^.data<x)and(p^.next<>nil)do
    begin
      q:=p;p:=p^.next
    end;
    if p^.data>=x{不是插在原尾结点之后}
    then begin
        if head=p then head:=po
        else q^.next:=po;
        po^.next:=p
      end
  else begin
        po^.next:=po;
        po^.next:=nil
```

```
    end;
  end;
end;
```

2、结点的删除

如下图所示，要在删除结点 P 的操作如下:



要删除结点 P,则只要将其前趋结点的指针域指向 P 的后继结点即可。

```
q^.next:=p^.next;
dispose(p);
```

例 9 将链表 head 中值为 X 的第一个结点删除

分析: 有三种情况存在:头结点的值为 X; 除头结点外的某个结点值为 X;无值为 X 的结点。为将前两种情况统一起来， 我们在头结点之前添加一个值不为 X 的哨兵结点。

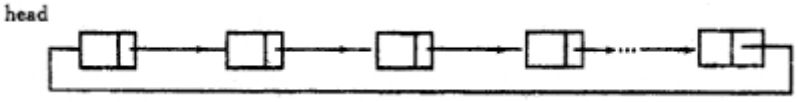
算法分两步：查找、删除。

过程如下：

```
procedure deleteing(var head:pointer;x:integer);
var p,q:pointer;
begin
  New(p);p^.data:=x-1;p^.next:=head;
  head:=p;{以上为添加哨兵结点}
  while(x<>p^.data)and(p^.next<>nil)do
  begin
    q:=p;
    p:=p^.next
  end;
  if x=p^.data{存在值为 X 的结点}
  then q^.next:=p^.next
  else writeln('NOt found!');
  head:=head^.next{删除哨兵}
end;
```

（四）环形链表结构

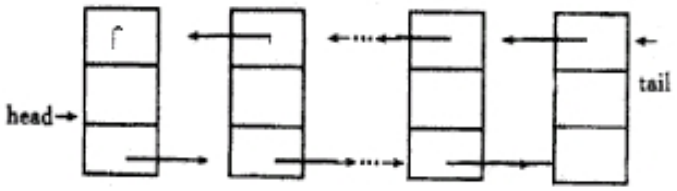
在单向链表中，表尾结点的指针为空。如果让表尾结点的指针域指向表头结点，则称为单向环形链表，简称单链环。如图所示。



单链环示意图

（五）双向链表结构

单链表中，每个结点只有一个指向其后继结点的指针域。如果每个结点不仅有一个指向其后继结点的指针域，还有一个指向其前趋的指针域，则这种链表称为双向链表。如图所示。



双向链表示意图

可用如下定义一个数据域为整型的双向链表:

```
    type pointer=^node;
        node=record
```

```
              prev:pointer;
              data:integer;
              next:pointer;
            end;
```

对双向链表的插入、删除特别方便。与单向链环相似，我们还可定义双向链环。

三、综合例析

例 10 读入一串以"#"为结束标志的字符，统计每个字符出现的次数。

分析：设置一个链表存放，每读入一个字符，就从链表的头结点向后扫描链表，如果在链表中此字符已存在，则其频率加 1， 否则将该字符的结点作为链表的新头结点，相应频率为 1。

源程序如下：

```
program ex11_10;
type ref=^letters;
      letters=record
              key:char;
              count:integer;
              next:ref;
            end;
  var k:char;
    sentinel,head:ref;
  procedure search(x:char);
  var w:ref;
  begin
    w:=head;
    sentinel^.key:=x;
    while w^.key<>x do w:=w^.next;
    if w<>sentinel
    then w^.count:=w^.count+1
    else begin
            w:=head;new(head);
            with head^ do
              begin
                  key:=x;count:=1;next:=w;
          end
end;
end;{of search}
procedure printlist(w:ref);
  begin
    while w<>sentinel do
      begin
        writeln(w^.key:2,w^.count:10);
        w:=w^.next;
      end;
  end;{of printlist}
begin{main program}
  new(sentine);
  with sentinel^ do
    begin
      key:='#';count:=0;next:=nil;
    end;
  head:=sentinel;
  read(k);
  while k<>'#' do
    begin
```

```
      search(k);read(k);
    end;
  printlist(head);
end.
```

例 11　用链表重写筛法求 2～100 之间所有素数程序。

源程序如下：

```
program ex11_12;
  uses crt;
  type link=^code;
       code=record
              key:integer;
              next:link;
            end;
  var head:link;
  procedure printlist(h:link);{打印链表 h}
    var p:link;
    begin
      p:=h;
      while p<>nil do
        begin
          write(p^.key,'-->');
          p:=p^.next;
        end;
    end;
  procedure buildlink;{建立链表}
    var p,q:link;
    i:integer;
    begin
      new(head);
      head^.key:=2;
      p:=head;
      for i:=3 to 100 do
        begin
          new(q);
          q^.key:=i;
          q^.next:=nil;
          p^.next:=q;
          p:=q;
        end;
    end;
  procedure prime;{筛法将找到的素数的倍数从链表中删除}
    var h,p,q:link;
    begin
      h:=head;
      while h<>nil do
        begin
          p:=h;q:=p^.next;
          while q<>nil do
            if (q^.key mod h^.key=0) then
              begin
                p^.next:=q^.next;
                dispose(q);
                q:=p^.next;
```

```
                end
              else begin
                     p:=q;
                     q:=q^.next;
                   end;
            h:=h^.next;
          end;
      end;
    begin{main program}
       clrscr;
       buildlink;
       printlist(head);
       writeln;
       prime;
       printlist(head);
     end.
```

练习

1、围绕着山顶有 10 个洞，一只兔子和一只狐狸各住一个洞，狐狸总想吃掉兔子。一天兔子对狐狸说，你想吃我有一个条件，你先把洞编号 1 到 10。你从第 10 号洞出发，先到第 1 号洞找我，第二次隔一个洞找我，第三次隔两个洞找我，以后依次类推，次数不限。若能找到我，你就可以饱餐一顿，在没找到我之前不能停止。狐狸一想只有 10 个洞，寻找的次数又不限，哪有找不到的道理，就答应了条件。结果就是没找着。

利用单链环编程，假定狐狸找了 1 0 0 0 次，兔子躲在哪个洞里才安全。

2、某医院病房的订位管理中， 将病人的记录按姓名的字母顺序排成一个链表。试编写程序，从键盘上输入下列字母，就可对病员记录进行管理：

（1） i ————新病人入院（插入一个病员记录）。

（2） d ————病人出院（删除一个病员记录，并显示该记录）。

（3） s ————查询某病员记录（显示该记录或"未找到"）。

（4） q ————在屏幕上列出所有的病员记录并结束程序。

3、编写一个简单的大学生新生入校登记表处理程序。

第十一章 类型文件

按数据的二进制代码形式存放时的文件称为类型文件。如果再按照组成类型文件的元素数据结构分，又可以分为有类型文件和无类型文件。其定义为：

```
type 类型名=file of 基类型；{有类型文件}
     类型名=file；          {无类型文件}
```

例如：var f:file of integer;

说明 f 为名的变量对应文件将用于存放整数。

var g:file;

说明 g 为名的变量对应文件的数据无任何规定。

Turbo Pascal 有关类型文件的函数和过程

（1）assign 过程

形式：assign(f,str);

功能：将文件名字符串 str 赋给文件变量 f，程序对文件变量 f 的操作代替对文件 str 的操作。

（2）rewrite 过程

形式：rewrite(f);

功能：建立并打开一个新的允许写磁盘文件，其文件名必须先由 assign 过程赋给变量 f。这时，指向文件元素的指针指向第一个元素，rewrite 过程所建立的文件为空文件。

（3）reset 过程

形式：reset(f);

功能：打开一个已经存在的磁盘文件，其文件名必须先由 assign 过程赋给变量 f,该文件只能读，指向文件元素的指针指向第一个元素。

（4）read 过程

形式：read(f,var 表);

功能：从磁盘文件 f 中，将数据依次读到 var 表表示的各个变量中。

（5）write 过程

形式：write(f,var 表);

功能：将 var 表所表示的各个变量的值依次写到磁盘文件 f 上。

（6）close 过程

形式：close(f);

功能：关闭和 f 关联的磁盘文件，在写操作时自动产生一个文件结束标志。

（7）seek 过程

形式：seek(f,n);

功能：把文件指针移到 f 指明文件的第 n 个元素。

（8）eof 函数

形式：eof(f);

功能：若文件指向文件尾，则返回 true,否则返回 false。

对有类型文件的写操作步骤为：

```
assign(f,str);
rewrite(f);
write(f,var 表);
close(f);
```

对有类型文件的读操作步骤为：

```
assign(f,str);
reset(f);
read(f,var 表);
close(f);
```

例：在磁盘上建立一个 1～50 的平方数的数据文件 zhoufei.dat。要求以一个数，这个数的平方数的格式写入。

```
program zhoufei;
  var f:file of integer;
  i:integer;
begin
  assign(f,'zhoufei.dat');
  rewrite(f);
  for i:=1 to 50 do write(f,i,sqr(i));
  close(f)
end.
```

文本文件

文本文件的内容有 ASCII 字符集中的字符组成，因此文本文件也称 ASCII 码文件，它可以用 DOS 中的 type 命令列出内容。文本文件具体是由一系列行组成，每一行可以包括 0 个或多个字符型成分，并以也行结束符结尾，文本文件类型 TXT 和类型文件 file of char 区别在于后者不包含行结束符。

文本文件和类型文件在读写上的差别在于前者只能按次序顺序读写，而后者可以不按照次序读写。适用文本文件的函数和过程除了用于类型文件操作的过程和函数外主要还有：

(1)readln 过程

形式：readln(f,var 表);或 readln(f);

功能：从磁盘文件 f 中，将数据依次读到 var 表表示的各变量中（其中 readln(f)只读数据），并将文件指针移到行结束符后，就是下一行开头。

（2）writeln 过程

形式：writeln(f,var 表)或 writeln(f);

功能：将 var 表所表示的各个变量的值依次写到磁盘文件 f 上去（writeln(f)不写值），然后再写一个行结束符。

（3）append 过程

形式：append(f);

功能：打开一个已经存在的磁盘文件，其文件名必须和 assign 过程中的变量名 f 相对应，该文件只能写，此时文件指针指向文件尾。

（4）eoln 函数

形式：eoln(f);

功能：若文件指针指向行结束符或文件结束符，则返回 true，否则返回 false。

对文本文件的写操作步骤：

```
assign(f,str);
rewrite(f);    或 append(f);
write(f,var 表);或 writeln(f);
close(f);
```

对文本文件的读操作步骤：

```
assign(f,str);
reset(f);
readln(f,var 表);或 readln(f);
close(f);
```

例：随机产生 30 个随机整数存放于文本文件 zhoufei.txt 中

```
program zhoufei;
  const n=30;
  var ra:text;
      i:integer;
begin
  randomize;
  assign(ra,'zhoufei,txt');
  rewrite(ra);
  for i:=1 to n do writeln(ra,random(100));
  close(ra)
end.
```

在 DOS 操作中，我们所谈及的文件称之为外部文件。外部文件是存储在外部设备上， 如：外存储器上，可由计算机操作系统进行管理，如用 dir、type 等命令直接对文件进行操作。

Pascal 所谈及的文件，称之为内部文件。内部文件的特点是文件的实体（实际文件）也是存储在外存储器上，成为外部文件的一分子，但在使用时必须在程序内部以一定的语句与实际文件联系起来，建立一一对应的关系，用内部文件的逻辑名对实际文件进行操作。内部文件的逻辑名必须符合ＰＡＳＣＡＬ语言标识符的取名规则。

Pascal 中的文件主要用于存放大量的数据。如：成绩管理，原始数据很多，使用文件先将其存入磁盘，通过程序读出文件中的数据再进行处理，比不使用文件要来得方便、有效。

Pascal 中的一个文件定义为同一类型的元素组成的线性序列。文件中的各个元素按一定顺序排列，可以从头至尾访问每一个元素，从定义上看，文件与数组相似，但它们之间有着明显不同的特征，主要表现在：

（１）文件的每一个元素顺序存贮于外部文件设备上（如磁盘上）。因此文件可以在程序进行前由 Pascal 程序或用文字编辑软件，如 edit、ws、Turbo Pascal 的 edit 命令等产生，或在运行过程中由程序产生，且运行完后，依然存贮在外部设备上。

（２）在系统内部，通过文件指针来管理对文件的访问。文件指针是一个保存程序在文件中位置踪迹的计算器，在一固定时刻，程序仅能对文件中的一个元素进行读或写的操作，在向文件写入一个元素或从文件读取一个元素后，相应的文件指针就前进到下一元素位置。而数组是按下标访问。

（３）在文件类型定义中无需规定文件的长度即元素的个数，就是说元素的数据可动态改变，一个文件可以非常之大，包含许许多多元素，也可以没有任何元素，即为一个空文件。而数组的元素个数则是确定的。

使用文件大致有以下几个步骤；

（１）说明文件类型，定义文件标识符；

（２）建立内部文件与外部文件的联系；

（３）打开文件；

（４）对文件进行操作；

（５）关闭文件。

Turbo Pascal 将文件分为三类：文本文件（顺序）、有类型文件（顺序或随机）和无类型文件

（顺序或随机）。下面将介绍这些文件及其操作。

一、文本文件

文本文件又称为正文文件或行文文件，可供人们直接阅读，是人机通信的基本数据形式之一。文本文件可用文字编辑程序（如 DOS 的 edit 或 Turbo Pascal 的编辑命令 edit）直接建立、阅读和修改， 也可以由ＰＡＳＣＡＬ程序在运行过程中建立。

１、文本文件的定义：

文本文件的类型为 TEXT，它是由 ASCII 字符组成的，是 Pascal 提供的标准文件之一。标准文件 TEXT 已由 Pascal 说明如下：

TYPE TEXT=FILE OF CHAR;

因此，TEXT 同标准类型 INTEGER、READ 等一样可以直接用于变量说明之中，无需再由用户说明。例如：

VAR F1，F2：TEXT;

这里定义了两个文本文件变量 F1 和 F2。

2、文本文件的建立

文本文件的建立有两种方法：

（１）直接用 Turbo Pascal 的 Edit 建立原始数据文件。

例 1　将下表中的数据存入名为 A.dat 的文件中。

3 4
29 30 50 60
80 90 70 75
60 50 70 45

操作步骤：

①进入 Turbo Pascal 的编辑状态；

②输入数据；

③存盘，文件名取 A.dat。

此时，已将数据存入文本文件 A.dat 中。文本文件也可用 DOS 中的 Edit 等软件建立。

（２）用程序的方式建立中间数据或结果数据文件。

用程序的方式建立文件操作步骤为：

①定义文本文件变量；

②把一外部文件名赋于文本文件变量，使该文本文件与一相应外部文件相关联；

命令格式：ASSIGN(f，name)

f 为定义的文本文件变量

name 为实际文件文件名

如：ASSIGN(F1,`FILEIN.DAT`)

或：ASSIGN(F1,`PAS\FILEIN.RES`)

这样在程序中对文本文件变量 F1 的操作，也就是对外部实际文件`FILEIN.DAT`或`FILEIN.RES`的操作。上例中文件`FILEIN.DAT`是存贮在当前目录中，而文件`FILEIN.RES`则是存贮在 PAS 子目录中。

③打开文本文件，准备写；

命令格式１：REWRITE(f)

功能：创建并打开新文件准备写，若已有同名文件则删除再创建

命令格式２：APPEND(f)

功能：打开已存在的文件并追加

④对文件进行写操作；

命令格式：WRITE(f，<项目名>)

或：WRITELN(f，<项目名>)

功能：将项目内容写入文件 f 中

⑤文件操作完毕后，关闭文件。

命令格式：CLOSE(f)

例 2 从键盘上读入表 12.1 的数据，用程序写入名为 B.dat 的文件中。

３、读取文本文件

文本文件内容读出操作步骤：

①定义文本文件变量；
②用 ASSIGN(f，name)命令，将内部文件 f 与实际文件 name 联系起来；
③打开文本文件，准备读；
命令格式：READ(f，<变量名表>) READLN(f，<变量名表>)
功能：读文件 f 中指针指向的数据于变量中
文本文件提供了另外两个命令，在文本的操作中很有用处，它们是：
EOLN(f)：回送行结束符
EOF(f)：回送文件结束符
⑤文件操作完毕，用 CLOSE(f)命令关闭文件。
例 3 读出例 12.1 建立的文本文件，并输出。

由于文本文件是以 ASCII 码的方式存储，故查看文本文件的内容是极为方便，在 DOS 状态可使用 DOS 中 TYPE 等命令进行查看，在 Turbo Pascal 中可以象取程序一样取出文件进行查看。

４、文本文件的特点

（１）行结构

文本文件由若干行组成，行与行之间用行结束标记隔开，文件末尾有一个文件结束标记。由于各行长度可能不同，所以无法计算出给定行在文本文件中的确定位置，从而只能顺序地处理文本文件，而且不能对一文本文件同时进行输入和输出。

（２）自动转换功能

文本文件的每一个元素均为字符型，但在将文件元素读入到一个变量（整型，实型或字符串型）中时，Pascal 会自动将其转换为与变量相同的数据类型。与此相反在将一个变量写入文本文件时，也会自动转移为字符型。

例４ 某学习小组有１０人，参加某次测验，考核 6 门功课， 统计每人的总分及各门的平均分，将原始数据及结果数据放入文本文件中。

分析

（１）利用 Turbo Pascal 的 EDIT 建立原始数据文件 TESTIN.DAT 存贮在磁盘中，其内容如下：

```
10 6
1 78 89 67 90 98 67
2 90 93 86 84 86 93
3 93 85 78 89 78 98
4 67 89 76 67 98 74
5 83 75 92 78 89 74
6 76 57 89 84 73 71
7 81 93 74 76 78 86
8 68 83 91 83 78 89
9 63 71 83 94 78 95
10 78 99 90 80 86 70
```

（２）程序读入原始数据文件，求每人的总分及各门的平均分；

（３）建立结果数据文件，文件名为 TEXTIN.RES.

程序：

例５ 读入一个行长不定的文本文件。排版，建立一个行长固定为６０个字符的文件， 排版要求：(1)当行末不是一个完整单词时，行最后一个字符位用'－'代替， 表示与下一行行头组成完整的单词；(2)第一行行头为两个空格，其余各行行头均不含有空格。

分析

（１）建立原始数据文件。

（２）程序边读入原始数据文件内容，边排版。

（３）每排完一行行长为６０字符，并符合题中排版条件，写入目标文件中。

设原始数据 TEXTCOPY.DAT 文件内容如下：

```
Pavel was arrested.
That dat Mother did not light the stove.
```

Evening came and a cold wind was blowing.
There was a knock at the window.
Then another.
Mother was used to such knocks,but this time she gave a little start of joy.
Throwing a shawl over her shoulders,she opened the door.
程序：

对 TEXTCOPY.DAT 文本文件运行程序得到排版结果文件 TEXTCOPY.RES 内容如下：

```
    Pavel was arrested.That dat Mother did not light the stov-
evening came and a cold wind was blowing.There was a knock
at the window.Then another.Mother was used to such knocks,b-
ut this time she gave a little start of joy.Throwing a shawl
over her shoulders,she opened the door.
```

二、有类型文件

文本文件的元素均为字型符。若要在文件中存贮混合型数据，必须使用有类型文件。

１、有类型文件的定义

有类型文件中的元素可以是混合型的，并以二进制格式存贮，因此有类型文件（除了字符类型文件，因为它实质上是文本文件）不象文本文件那样可以用编辑软件等进行阅读和处理。

有类型文件的类型说明的格式为：

类型标识符＝File of 基类型；

其中基类型可以是除了文件类型外的任何类型。例如：

```
FILE1=FILE OF INTEGER;
FILE2=FILE OF ARRAY[1--10] OF STRING;
FILE3=FILE OF SET OF CHAR;
FILE4=FILE OF REAL;
FILE5=FILE OF RECORD;
NAME:STRING;
COURSE:ARRAY[1--10] OF READ;
SUN:READ;
END;
```

等等，其中 FILE2,FILE3,FILE5 中的数组、集合、记录等类型可以先说明再来定义文件变量。

例如：

```
VAR
F1:FILE;
F2,F3:FILE3;
F4:FILE5;
```

与前面所有类型说明和变量定义一样，文件类型说明和变量定义也可以合并在一起，例如：

```
VAR
F1:FILE OF INTEGER;
F2，F3:FILE OF SET OF CHAR;
F4:FILE OF RECORD
NAME:STRING;
COURSE:ARRAY[1--10] OF REAL;
SUM:READ;
END;
```

Turbo Pascal 对有类型文件的访问既可以顺序方式也可以用随机方式。

为了能随机访问有类型文件，Turbo Pascal 提供如下几个命令：

命令格式１：seek(f,n)

功能：移动当前指针到指定 f 文件的第 n 个分量，f 为非文本文件，n 为长整型

命令格式２：filepos(f)

功能：回送当前文件指针，当前文件指针在文件头时，返回，函数值为长整型

命令格式 3 ：filesize(f)

功能：回送文件长度，如文件空，则返回零，函数值为长整型

２、有类型文件的建立

有类型文件的建立只能通过程序的方式进行，其操作步骤与文本文件程序方式建立的步骤相仿，不同之处：（１）有类型文件的定义与文本文件的定义不同；（２）有类型文件可以利用 SEEK 命令指定指针随机写入。

３、有类型文件的访问

有类型文件访问的操作步骤与文本文件的程序访问操作步骤相仿，区别之处：（１）有类型文件的定义与文本文件的定义不同；（２）有类型文件可以利用 SEEK 命令访问文件记录中的任一记录与记录中的任一元素。

例 6　建立几个学生的姓名序、座号、六门课程成绩总分的有类型文件。

分析：为简单起见，这里假设已有一文本文件 FILEDATA.TXT，其内容如下：

```
10
li hong
1 89 67 56 98 76 45
wang ming
2 99 87 98 96 95 84
zhang yi hong
3 78 69 68 69 91 81
chang hong
4 81 93 82 93 75 76
lin xing
5 78 65 90 79 89 90
luo ze
6 96 85 76 68 69 91
lin jin jin
7 86 81 72 74 95 96
wang zheng
8 92 84 78 89 75 97
mao ling
9 84 86 92 86 69 89
cheng yi
10 86 94 81 94 86 87
```

第一个数 1 0 表示有 1 0 个学生，紧接着是第一个学生的姓名、座号、 6 科成绩，然后是第二个学生，等等。

从文本文件中读出数据，求出各人的总分，建立有类型文件，设文件名为 filedata.fil，文件的类型为记录 studreco，见下例程序。

程序：

例 7　产生数 1 － 1 6 的平方、立方、四次方表存入有类型文件中， 并用顺序的方式访问一遍，用随机方式访问文件中的 1 1 和 1 5 两数及相应的平方、立方、四次方值。

分析：建立有类型文件文件名为 BIAO.FIL，文件的类型为实数型。

（１）产生数 1 － 1 6 及其平方、立方、四次方值，写入 BIAO.FIL，并顺序读出输出；

（２）用 SEEK 指针分别指向 1 1 和 1 5 数所在文件的位置，其位置数分别为 1 0 × 4 和 1 4 × 4 （注意文件的第一个位置是 0 ），读出其值及相应的平方、立方、四次方值输出。

程序：

程序运行结果如下：

另外，Turbo Pascal 还提供了第三种形式文件即无类型文件，无类型文件是低层 I／O 通道，如果不考虑有类型文件、 文本文件等存在磁盘上字节序列的逻辑解释，则数据的物理存储只不过是一些字节序列。这样它就与内存的物理单元一一对应。无类型文件用 1 2 8 个连续的字节做为一个记录（或分量）进行输入输出操作，数据直接在磁盘文件和变量之间传输，省去了文件缓解区，因此比其它文件少占内存，主要用来直接访问固定长元素的任意磁盘文件。

无类型文件的具体操作在这里就不一一介绍，请参看有关的书籍。

三、综合例析

例 8　建立城市飞机往返邻接表。文本文件 CITY.DAT 内容如下：

第一行两个数字Ｎ和Ｖ；

Ｎ代表可以被访问的城市数，Ｎ是正数＜１００；

Ｖ代表下面要列出的直飞航线数，Ｖ是正数＜１００；

接下来Ｎ行是一个个城市名，可乘飞机访问这些城市；

接下来Ｖ行是每行有两个城市，两城市中间用空格隔开，表示这两个城市具有直通航线。

如：CITY1 CITY2 表示乘飞机从 CITY1 到 CITY2 或从 CITY2 到 CITY1。

生成文件 CITY.RES，由 0 、 1 组成的Ｎ×Ｎ邻接表。

邻接表定义为：

分析

（１）用从文本文件 city.dat 中读入Ｎ个城市名存入一些数组 CT 中；

（２）读入Ｖ行互通航班城市名，每读一行，查找两城市在 CT 中的位置 L、K，建立邻接关系，lj[l,k]=1 和 lj[k,j]=1；

（３）将生成的邻接表写入文本文件 CITY.RES 中。

设 CITY.DAT 内容如下：

```
10 20
fuzhou
beijin
shanghai
wuhan
hongkong
tiangjin
shenyan
nanchan
chansa
guangzhou
fuzhou beijin
fuzhou shanghai
fuzhou guangzhou
beijin shanghai
guangzhou beijin
wuhan fuzhou
shanghai guangzhou
hongkong beijin
fuzhou hongkong
nanchan beijin
nanchan tiangjin
tiangjin beijin
chansa shanghai
guangzhou wuhan
chansa beijin
wuhan beijin
shenyan beijin
shenyan tiangjin
```

shenyan shanghai
shenyan guangzhou
程序：

得到CITY.RES文件内容如下：
10
1 fuzhou
2 beijin
3 shanghai
4 wuhan
5 hongkong
6 tiangjin
7 shenyan
8 nanchan
9 chansa
10 guangzhou
0 1 1 1 1 0 0 0 0 1
1 0 1 1 1 1 1 1 1 1
1 1 0 0 0 0 1 0 1 1
1 1 0 0 0 0 0 0 0 1
1 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 1 0 0
0 1 1 0 0 1 0 0 0 1
0 1 0 0 0 1 0 0 0 0
0 1 1 0 0 0 0 0 0 0
1 1 1 1 0 0 1 0 0 0
例9 对例1 2.3的FILEDATE.FIL文件内容按总分的高低顺序排序。
分析：
文件的排序就是将文本文件的各分量按一定要求排列使文件有序，文件排序有内排序和外排序二种，内排序是指将文件各分量存入一个数组，再对数组排列，最后将该数组存入原来的文件。外排列不同于内排列，它不是将文件分量存入数组，而是对文件直接排序，内排序比外排序速度要快，但当文件很大时，无法调入内存，此时用外排序法较合适。
本程序使用过程SEEK，实现外排序。
程序：

习 题

1、编一程序，计算文本文件中行结束标志的数目。
2、计算文本文件的行长度的平均值、最大值和最小值。
3、一文本文件FILE.DAT存放N个学生某学科成绩，将成绩转换成直方图存入FILE.RES文件中。
如FILE.DAT内容为:
5
78 90 87 73 84
得到直方图文件FILE.RES内容为:
5
****
****
****
****
****

4、银行账目文件含有每一开户的账目细节：开户号、姓名、地址、收支平衡额。写一程序，读入每一开户的账目，生成银行账目文件。

5、通讯录文件每个记录内容为：姓名、住址、单位、邮编、电话，编一程序按姓名顺序建立通讯录文件，要求先建立文件，再对文件按姓名顺序进行外排序。

程序的调试

任何一个天才都不敢说，他编的程序是100%正确的。几乎每一个稍微复杂一点的程序都必须经过反复的调试，修改，最终才完成。所以说，程序的调试是编程中的一项重要技术。我们现在就来掌握一下基本的程序调试。 我们以下的示范，是以时下比较流行的Borland Pascal 7.0为例子，其他的编程环境可能略有不同，但大致上是一致的。

我们先编一个比较简单的程序，看看程序是如何调试的。

```
program tiaoshi;
var i:integer;
begin
for i:=1 to 300 do
begin
if i mod 2 = 0 then
if i mod 3 = 0 then
if i mod 5 = 0 then
writeln(i);
end;
end.
```

该程序是输出300以内同时能被2，3，5整除的整数。 现在我们开始调试。 调试有多种方法，先介绍一种，权且叫步骤法，步骤法就是模拟计算机的运算，把程序每一步执行的情况都反映出来。通常，我们有F8即STEP这个功能来实现，如图： 不断地按F8，计算机就会一步步地执行程序，直到执行到最后的“end.”为止。

可能你还没有发现F8的威力，我们不妨把上面的程序略微修改一下，再配合另外的一种调试的利器watch，你就会发现步骤法的用处。

```
program tiaoshi;
var i:integer;
a,b,c:boolean;
begin
for i:=1 to 300 do
begin
a:=false;
b:=false;
c:=false;
if i mod 2 = 0 then a:=true;
if i mod 3 = 0 then b:=true;
if i mod 5 = 0 then c:=true;
if a and b and c then writeln(i);
end;
end.
```

如图，我们单击菜单栏中debug选项，里面有一项叫watch的选项，我们单击它。

就会出现一个watch窗口：

watch窗口可以让我们观察变量的变化情况，具体操作是在watches窗口内按Insert键：

这时，屏幕上弹出一个菜单，我们输入所需要观察的变量名，我们分别输入i,a,b,c这4个变量名，于是watches窗口内就有如下的4个变量的状态：

这时，我们再次使用步骤法，我们会发现，这4个变量的状态随着程序的执行而不断变化，比如：

这样我们就可以方便地知道执行每一步之后，程序的各个变量的变化情况，从中我们可以知道我们的程序是否出错，在哪里出错，方便我们及时地修改。 下一次，我们介绍另外的一种方法，断点法。

程序的调试(二)

在前面我们已经学习了基本的程序调试方法——步骤法。步骤法有一个缺点，就是在遇到循环次数比较多或者语句比较多的时候，用起来比较费时，今天我们来学习一种新的也是常用的调试方法——断点法。
所谓断点法，就是在程序执行到某一行的时候，计算机自动停止运行，并保留这时各变量的状态，方便我们检查，校对。我们还是以前面求同时能被 2，3，5 整除的 3000 以内的自然数为例，具体操作如下：
我们把光标移动到程序的第 14 行，按下 ctrl+F8，这时我们会发现，该行变成红色，这表明该行已经被设置成断点行，当我们每次运行到第 14 行的时候，计算机都会自动停下来供我们调试。
我们必须学以致用，赶快运用刚学的 watch 方法，看看这家伙到底有多厉害。
请记住，计算机是执行到断点行之前的一行，断点行并没有执行，所以这时 b:=true 这一句并没有执行。
断点行除了有以上用处之外，还有另外一个重要用处。它方便我们判断某个语句有没有执行或者是不是在正确的时刻执行，因为有时程序由于人为的疏忽，可能在循环或者递归时出现我们无法预料的混乱，这时候通过断点法，我们就能够判断程序是不是依照我们预期的顺序执行。
行了，你毕业了，程序的调试你就算是基本过关了。有什么问题请发信到 junhuizh@163.net 此邮件地址受 spam bots 保护，需要使用 Javascript 功能来查阅。 和我讨论。发个奖给你。

Pascal 运行错误对照表

运行错误是指程序运行时出现的错误，当发生时，Turbo Pascal 显示如下信息：
RUNTIME ERROR NNNN AT XXXX：YYYY
其中，nnnn 是运行错误代码，xxxx 是错误发生的程序段，yyyy 是错误地址偏移。

DOS 错误代码：
1 无效 DoS 功能号
2 文件末找到
3 路径未找到
4 打开文件过多
5 禁止文件存取
6 无效文件句柄
12 无效文件存取代码
15 无效驱动器号
16 不能删除当前日录
17 不能跨驱动器改文件名

I/O 错误
100 磁盘读错误
101 磁盘写错误
102 文件变量末赋值
103 文件未打开
104 文件未用输入方式打开
105 文件末用输出方式打开
106 无效数字格式

严重错误
150 磁盘写保护
151 未知单元
152 驱动器未准备好
153 未知命令
154 数据 CRC 校验错
155 驱动器请求的结构长度错
156 磁盘定位错
157 未知媒介类型

158 扇区末找到
159 打印机缺纸
160 设备写失败
161 设备读失败
162 硬件故障

致命错误
200 被零除
201 范围检查错
202 堆栈溢出错
203 堆溢出错
204 无效指针操作
205 浮点上溢出
206 浮点下溢出
207 无效浮点运算
208 未安装覆盖管理程序
209 覆盖文件读错
210 对象未初始化
211 调用抽象方法
212 流登计错
213 集合下标越界
214 集合溢出
215 算术上溢错误
216 存取非法
217 控制-C
218 授权指令
219 无效的 TYPECAST
220 无效的变体 TYPECAST
221 无效的变体操作
222 没有变体方法调用 DISPATCHER
223 不能建立变体数组
224 变体不包含数组
225 变体数组边界错误
226 TLS 初始化错误

Pascal 编译错误对照表

下面列出在编译程序时可能出现的错误，在集成环境下，Turbo Pascal 将自动加载源程序并定位于出错处。
1 内存溢出
2 缺标识符
3 标识符未定义
4 标识符重定义
5 语法错误
6 实型常量错
7 整型常量错
8 字符串常量跨行
9 文件嵌套过多
10 非正常文件结束
11 行过长
12 缺类型标识符
13 打开文件过多

14 无效文件名
15 文件未找到
16 磁盘满
17 无效编译指示
18 文件过多
19 指针定义中未定义类型
20 缺变量标识符
21 类型错误
22 结构过长
24 文件分量不能为文件
25 无效字符串长度
26 类型不匹配
27 无效子界基类型
28 下界大于上界
29 缺有序类型
30 缺整型常数
31 缺常数
32 缺整型或实型常数
33 缺指针类型标识符
34 无效的函数结果类型
35 缺标号标识符
36 缺 BEGIN
37 缺 END
38 缺整型表达式
39 缺有序表达式
40 缺布尔表达式
41 操作数类型与操作符不匹配
42 表达式错
43 非法赋值
44 缺字段标识符
45 目标文件过长
46 未定义外部标识符
47 无效*.OBJ 文件记录
48 代码段过长
49 数据段过长
50 缺 DO
51 无效 PUBLIC 定义
52 无效 EXTRN 定义
53EXTRN 定义过多
54 缺 OF
55 缺 INTERFACE
56 无效重定位引用
57 缺 THEN
58 缺 TO 或 DOWNTO
59 未定义的向前引用
60 过程过多
61 无效类型转换
62 被零除 D
63 无效文件类型
64 不能读写该类型的变量
65 缺指针变量
66 缺字符串变量
67 缺字符串表达式

68 单元循环引用
69 单元名不匹配
70 单元版本不匹配
71 单元重名
72 单元文件格式错误
73 缺 IMPLEMENTATlON
74 常数与 CASE 类型不相匹配
75 缺记录变量
76 常数越界
77 缺文件变量
78 缺指针变量
79 缺整型或实型表达式
80 标号不在当前块中
81 标号已定义
82 标号未定义
83 无效参数
84 缺 UNIT
85 缺“;”
86 缺“:”
87 缺“,”
88 缺“(”
89 缺“)”
90 缺“＝”
91 缺“：＝”
92 缺“[”或“(.”
93 缺“)”或“.)”
94 缺“.”
96 变量过多
97 无效 FOR 控制变量
98 缺整型变量
99 此处不允许用文件和
100 字符串长度不匹配
101 无效字顺序
102 缺字符串常数
103 缺整型或实型变量
104 缺有序变量
105INLINE 错
106 缺字符表达式
107 重定位项过多
112CASE 常量越界
113 语句错
114 不能调用中断过程
116 必须在 8087 方式下编译
117 末找到目标地址
118 此处不允许包含文件
120 缺 NIL
121 无效限定符
122 无效变量引用
123 符号过多
124 语句部分过长
126 文件必须为变量参数
127 条件符号过多
128 条件指令错位

130 初始条件定义错
131 过程和函数头与前面定义的不匹酉
132 严重磁盘错误
133 不能计算该表达式
134 表达式错误结束
135 无效格式说明符
136 无效间接引用
137 此处不允许结构变量
138 无 SYSTEM 单元不能计算
139 不能存取该符号
140 无效浮点运算
141 不能将覆盖编译至内存
142 缺过程和函数变量
143 无效过程或函数引用
144 不能覆盖该单元
147 缺对象类型
148 不允许局部对象类型
149 缺 VIRTUAL
150 缺方法标识符
151 不允许虚拟构造方法
152 缺构造方法标识符
153 缺释放方法标识符
154FAIL 只允许在构造方法内使用
155 无效的操作符和操作数组合
156 缺内存引用
157 不能加减可重定位符号
158 无效寄存器组合
159 未激活 286 / 287 指令
160 无效符号引用
161 代码生成错
162 缺 ASM